NTech
Lehrstuhl für
Nukleartechnik

Technische Universität München

Master Thesis

by Adrià Carrasco Boix
Matrikelnummer 03639904

Application of the Lattice Boltzmann Method to Issues of Coolant Flows in Nuclear Power Reactors

Betreuer TUM: Prof. Dr. Rafael Macián-Juan
Betreuer : Dr. rer. nat. Martin Ohlerich
Ausgegeben: Mai 2013
Abgegeben: Oktober 2013

ETSEIB

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Declaration

I hereby declare that I have made this thesis independently and without the help of others. Ideas and quotes that I have taken over directly or indirectly from other sources are marked as such. This thesis has not been submitted to any inspection authority in the same or similar form and has not been published.

I hereby agree that this thesis can be made available to the public by the *Lehrstuhl für Nukleartechnik*.

München, 7^{th} October 2013

Adrià Carrasco Boix

Aknowledgements

This thesis could not be finished without the help and support of many people, who are greatfully aknowledged here.

First of all, I want to express my deepest gratitude to my supervisor *Dr. rer. nat* Martin Ohlerich. This thesis could not be written without his guidance and the values discussions that we had. Also I would like thank its constantly encouragement and trust without hesitation.
I would like to express my apreciation to *Prof. Dr.* Rafael Macián-Juan and to the *Lehrstuhl für Nukleartechnik* for the ease given to me to develop my thesis in this department.

I would like to thank the incalculable support of many people.
Primerament, voldria agrair a mons pares el seu constant suport durant tots aquests anys, i pel sacrifici que han fet. Moltes gràcies, de debò. Sense vosaltres no ho hauria pogut conseguir.
A l'Alba, per sempre fer-m'ho tot més fàcil i pel suport i l'amor incondicional que cada dia em dóna.
Al Frans, Arian i Weiss, pels anys viscuts a Barcelona.
Al Guille i el Pere, per ser persones que sempre m'han donat força.
Als amics de Reus i Barcelona per estar sempre allí.
Finalmente, y no por eso menos importante, a todos mis amigos de Munich.

Contents

List of Figures

List of Algorithms

List of Acronyms

CFD Computational Fluid Dynamics

LBM Lattice Boltzmann Method

NS Navier-Stokes

BGK Bhatnagar, Gross and Krook

EOS Equation Of State

PWR Pressurized Water Reactor

List of Notations

ν_{lb}	Viscosity in the lattice framework
ρ	Density
τ	Relaxation time
$\vec{e}_a$	Vector for the direction a in the lattice framework
$\vec{Q}$	Force corrector term
c_s	Speed of sound
E	Energy
f	Distribution function
f_a	Discrete distribution function in the velocity direction a
f_a^{eq}	Discrete equilibrium distribution function in the velocity direction a
k	Boltzmann Constant
m	Mass
P	Pressure
Re	Reynolds number
T	Temperature
w_a	Weight factor for the direction a

Abstract

In the past years, the Lattice Boltzmann Method (LBM) has been widely used by the scientific community as an alternative to the conventional numerical solvers for the Navier-Stokes (NS) equations. The present work in this thesis aims at studying the LBM for fluid dynamics. The main topics are concentrated in three aspects: the description of the model, the validation of this model, and the application of this model to an engineering case.
In the first part the model is defined. Therefore, the Boltzmann equation and the Boltzmann distribution are defined. Also, the explanation of the framework where this method works is included. Then, the *BGK-Approximation* and the discretization of the Boltzmann equation are introduced. The algorithm needed to apply the model will also be explained together with numerical stability issues that one must take into account when it is implemented. An explanation of the different boundary conditions will also be summarized.
In chapter 3 the LBM will be applied to different 3D cases to test its accuracy and validate the model. The Poiseuille and Couette flow will be studied and compared analytically. Lid driven cavity and the flow around an obstacle will also be simulated.
In chapter 4, after the model has been validated, the LBM is used to simulate a complex situation to simulate the flow pattern in a *lower plenum of a PWR reactor core*, taking into account several simplifications, to understand the possibilities of the LBM implemented.
To be able to perform chapter 3 and 4 an implementation in `C++` has been developed.

Chapter 1

Introduction

The LBM is a class of Computational Fluid Dynamics (CFD) methods for the simulation of the fluid flow pattern behaviour. Nowadays, the vast majority of the CFD tools are based on method for solving the NS equations, but an alternative approach to these CFD methods was introduced in the late 1980s with the lattice gas automata. Historically, lattice gas methods can be considered as the predecessors of the LBM. These were models in which every particle was allowed to move on a discrete lattice and conserved the mass and momentum through the local collisions [Bui97]. It was shown that these methods follow a continuity equation, which is but an expression of the continuum hypothesis, whereas the local collisions are indeed a single particle description. Therefore, they were thought to be capable to simulate hydrodynamics problems. However lattice gas methods suffered from statistical noise and the difficulty of the model to handle three dimensional domains, thus, these limited their success.

Later, a different approach was considered. This approach consists in interpreting the single motions between lattice points as those of single particles, and form a density out of the number of particles confined in a certain lattice volume. This density corresponds to a local averaging procedure to obtain the continuity.

This migration from the single particle picture to a continuum density picture corresponds to the migration from a statistical description (where every particle is represented) to an averaged description. Therefore the statistical noise vanishes. This is one of the key points of the model. The LBM is motivated by the Boltzmann Equation description, and contains the solutions of the NS equations as approximation, since NS equations are derivable from the Boltzmann equation [Thu07].

When facing a problem in which there are simulations of transport equations, for example a simulation of a certain fluid flowing through a certain volume, one must to take care of the different possible approaches.
First of all, there is the macroscopic approach. In this approach the NS equations are needed to be solved. The main problem in solving these equations is the closure of the equations by using constitutive Equation Of State (EOS), which they are often only aprroximately or phenomenologically known. To solve these equations, first one has to discretize them and fix the boundary conditions. Then, these algebraic equations can be solved iteratively until its convergence. In this scale the value of the variables of the system (*i.e.* velocity, temperature, pressure...) is an average value of the values of these variables over the whole finite domain.
On the other hand, there is the microscopic description. On this scale, the medium is considered to be composed of small particles (*i.e.* atoms or molecules) and they can interact with each other by collisions. This an intuitive consideration, and there is only need to solve the ordinary differential equations of Newton's second law (momentum conservation). The problem with this approach is the large amount of equations that one needs to solve for each time step. For instance, the order of magnitude of the number of molecules in a small volume like a water drop it is about 10^{16} [Moh11]. At this scale the connection between the microscopic variables (*e.g.* positions and velocities) can be related to the macroscopic quantities (*i.e.* temperature and pressure) under certain conditions. One can easily see that the huge

amount of equations needed to be solved every time step for a reasonable large volume cannot be handled by the currently avaible computer capability.

Half way between the microscopic and macroscopic scale is located the LBM. This method works in the mesoscopic scale, where the system is described microscopically but treated statistically, and it solves the Boltzmann Equation in an approximate way to ensure the mass, momentum and energy conservation. It can be seen schematically in Fig. 1.1.

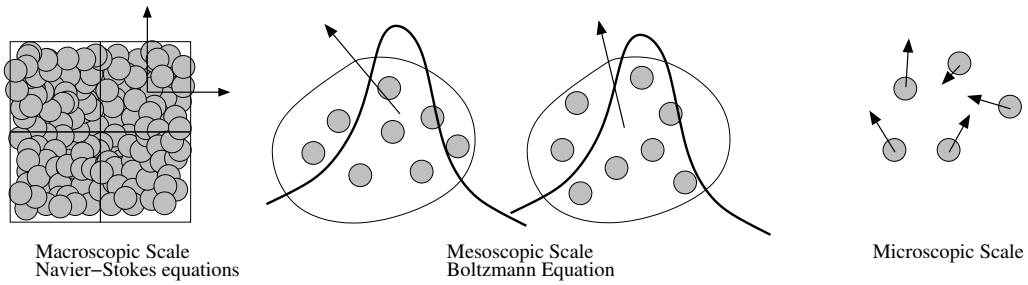


Figure 1.1: Different approaches in simulation techniques

As it will be shown later with the explanation of the Boltzmann equation, the main idea of Boltzmann's work is that gases are composed of particles interacting with each other that can be described by the laws of classical mechanics and they can be treated statistically to avoid working with so many equations. Then, the interaction between those particles could be treated with just notions of streaming and collisions in space.
Furthermore, a big advantage of the LBM is that it is highly parallelizable due to the fact that the streaming and collisions treatment is solved locally in every discrete point. In addition, in the LBM natural issues like multi-phase flows and non-equilibrium states can be incorporated. Therefore the interest of the scientific community for this method is in a constant increase.

Chapter 2

Description of the Model

2.1 The Boltzmann Distribution

The basic idea of the LBM is to treat statistically groups of particles to be able to consider them as a unit. This statistical treatment is made using the *Maxwell-Botlzmann distribution*, which is the equilibrium phase-space distribution [SJ06]. Boltzmann found that the probability for a system in a thermal equilibrium at temperature T, to be in a particular energetic state E is proportional to $e^{-E/(kT)}$ [Moh11], where k is the Boltzmann constant,

$$f(v_x, v_y, v_z) \; = \; Ae^{-\frac{E(v_x, v_y, v_z)}{kT}} \tag{2.1}$$

Then, it is known that the energy of a molecule is

$$E \; = \; \frac{1}{2}m(\vec{v})^2 = \frac{1}{2}m\left(\sqrt{v_x^2 + v_y^2 + v_z^2}\right)^2 \tag{2.2}$$

Since, mathematically,

$$f(v_x, v_y, v_z)dv_x dv_y dv_z \equiv f(\vec{v})d^3v$$

To know d^3v, a variable change must be done,

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = v \begin{pmatrix} sin\theta cos\varphi \\ sin\theta sin\varphi \\ cos\theta \end{pmatrix}$$

therefore, if Ω is the domain where all angles are contained,

$$d^3v \;=\; v^2 dv\, d\Omega$$

Then, if this is used,

$$f(\vec{v})d^3v \;=\; A\, e^{-\frac{v^2}{2kT/m}}\; v^2\, dv\, d\Omega$$

Hence, the distribution follows

$$f(v) = \int_\Omega f(\vec{v})d^3v \;=\; A\, e^{-\frac{v^2}{2kT/m}} v^2 dv \int_\Omega d\Omega$$

and it is known that,

$$\int_\Omega d\Omega \;=\; 4\pi$$

Thereby, the *Maxwell-Boltzmann distribution* is,

$$f(v) \;=\; A' v^2 e^{-\frac{v^2}{2kT/m}} \tag{2.3}$$

And a normalized probability distribution should have a value equal to one when it is integrated for all values of velocity possible

$$\int_{-\infty}^{\infty} A' e^{-\frac{mv^2}{2kT}}\, v^2\, dv \;=\; 1 \tag{2.4}$$

Therefore, if the integral is solved to know the A' constant

$$A' \;=\; 4\pi \left(\frac{m}{2\pi kT}\right)^{\frac{3}{2}} \tag{2.5}$$

If formulas (2.1), (2.2) and (2.5) are combined to reach the *Maxwell-Boltzmann distribution*:

$$f(v) \;=\; 4\pi \left(\frac{m}{2\pi kT}\right)^{3/2} v^2 e^{-\frac{mv^2}{2kT}} \tag{2.6}$$

If the velocity is a vector, the distribution function is

$$f(\vec{v}) \;=\; \left(\frac{m}{2\pi kT}\right)^{3/2} e^{-\frac{mv^2}{2kT}} \tag{2.7}$$

As the reader might have noticed the distribution function shown in formula (2.8) is slightly different from the *Maxwell-Boltzmann distribution function.* This is because this is the distribution function of $f(\vec{v})$ and not $f(v)$. Changing the latter to the first requires to integrate all angular degrees of freedom $4\pi\, v^2 dv$.

2.2 The Boltzmann Equation

The distribution $f(\vec{r}, \vec{v}, t) d^3r d^3v$ gives the probability of finding a single molecule inside a small volume d^3r, around the position $\vec{r}$ with a certain velocity $\vec{v}$ at time t. So in principle, the probability of finding a molecule with a velocity $\vec{v}$ between $\vec{v}$ and $\vec{v} + d\vec{v}$ and at a certain position between $\vec{r}$ and $\vec{r} + d\vec{r}$ at time t is given by the following relation [SJ06]

$$f(\,\vec{r} \,+\, d\vec{r},\, \vec{v} \,+\, d\vec{v},\, t) \quad = \quad f(\,\vec{r},\, \vec{v},\, t)\, d\vec{r}\, d\vec{v} \tag{2.8}$$

In the supposed situation where collisions do not exist, every molecule is able to move freely as they do not interact with other molecules. Then, a given external force $\vec{F}$ is introduced, which acts on a molecule of unit mass. Therefore the velocity $\vec{v}$ of the molecule will change to $\vec{v} + \vec{F}\, dt$, which can be expressed as $\vec{v} + \vec{F}\, dt \quad = \quad \vec{v} + (\frac{d\vec{v}}{dt})\, dt \quad = \quad \vec{v} + d\vec{v}$ and the position will change as well from $\vec{r}$ to $\vec{r} + \vec{v} dt$, which also can be simplified to $\vec{r} + \vec{v}\, dt \quad = \quad \vec{r} + (\frac{d\vec{r}}{dt})\, dt \quad = \quad \vec{r} + d\vec{r}$. Hence, since there are no collisions allowed, the distribution density before applying the external force

$\vec{F}$ and the distribution after a differential time step dt follows this relation [Moh11]:

$$f(\vec{r} + d\vec{r}, \vec{v} + d\vec{v}, t + dt)\, d\vec{r}\, d\vec{v} = f(\vec{r}, \vec{v}, t)\, d\vec{r}\, d\vec{v} \tag{2.9}$$

It is because molecules can move freely, that every molecule with a given initial state is free to arrive to a given final state after $\vec{F}$ is applied. This process can be seen in Fig. 2.1. As it will be explained later, Eq (2.9) represents the *Streaming Step*, which is one of the two basic steps of the LBM.

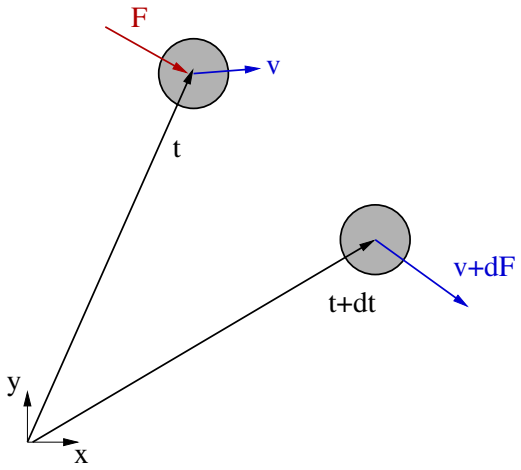


Figure 2.1: Effect of $\vec{F}$ on the particle velocity.

If collisions are considered, the relation (2.9) changes due to the fact that molecules are not able to move with complete freedom around the domain. Hence, a collision term must be added in the relation (2.9) to restrain the molecules freedom. This term is the rate of change between the number of molecules that are between $\vec{r} + d\vec{r}$ and $\vec{v} + d\vec{v}$ after a time differential dt.

The rate of change can be expressed as [Moh11]:

$$\frac{df}{dt} = \Omega \tag{2.10}$$

It is easier to explain this rate of change when the complete Boltzmann equation is known. Therefore, this is the Boltzmann equation in its differential form,

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial \vec{r}}\,\vec{v} \; + \frac{\vec{F}}{m}\frac{\partial f}{\partial \vec{v}} = \; \Omega \tag{2.11}$$

This equation expresses that the total rate of change in the distribution function is equal to the rate of collisions, plus the fluxes of particles entering and leaving the volume of study.
One can derive from Eq. (2.10) to Eq. (2.11) following this procedure.
Eq.(2.10) implies that the total rate of change of the distribution function is equal to the rate of collisions. Since $\vec{r}$, $\vec{v}$ and t are variables of the density distribution function f, the total rate of change is

$$df \; = \; \frac{\partial f}{\partial \vec{r}}d\vec{r} + \frac{\partial f}{\partial \vec{v}}d\vec{v} + \frac{\partial f}{\partial t}dt \tag{2.12}$$

and arranging this equation yields,

$$\frac{df}{dt} \; = \; \frac{\partial f}{\partial \vec{r}}\frac{d\vec{r}}{dt} + \frac{\partial f}{\partial \vec{v}}\frac{d\vec{v}}{dt} + \frac{\partial f}{\partial t} \tag{2.13}$$

if it is taken into account that $\vec{v} = \frac{d\vec{r}}{dt}$ and $\vec{a} = \frac{d\vec{v}}{dt}$, (2.13) can be written as

$$\frac{df}{dt} \; = \; \frac{\partial f}{\partial \vec{r}}\vec{v} + \frac{\partial f}{\partial \vec{v}}\vec{a} + \frac{\partial f}{\partial t} \tag{2.14}$$

It is known the relation between $\vec{a}$ and $\vec{F}$ through the *Newton*'s second law $\vec{a} = \frac{\vec{F}}{m}$. So (2.13) is

$$\frac{df}{dt} \; = \; \frac{\partial f}{\partial \vec{r}}\vec{v} + \frac{\partial f}{\partial \vec{v}}\frac{\vec{F}}{m} + \frac{\partial f}{\partial t} \tag{2.15}$$

Therefore, there is the complete Boltzmann equation

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial \vec{r}}\,\vec{v} \; + \frac{\vec{F}}{m}\frac{\partial f}{\partial \vec{v}} = \; \Omega \tag{2.16}$$

and in the vector form

$$\frac{\partial f}{\partial t} + \vec{v}\nabla f \; = \; \Omega \tag{2.17}$$

2.3 The BGK-Approximation.

It is not easy to solve the Boltzmann equation due to the collision term. Using the Chapman-Enskog expansion [Lut06], the collision term can be related to the viscosity when deriving the NS equation. An approximation can be made without introducing significant error to the result. This approximation was developed by Bhatnagar, Gross and Krook (BGK) in 1954 as a simplification of the collision operator.

$$\Omega = \omega \left(f^{eq} - f \right) = \frac{1}{\tau} \left(f^{eq} - f \right) \tag{2.18}$$

where ω is the collision frequency, and its inverse τ is the relaxation time [Moh11].The term f^{eq} denotes the distribution function value at the local equilibrium [Gom94].
In the lattice framework (defined in section 2.5), the relaxation time is related with the viscotisy in this framework, ν_{lb}, by the following formula [SJ06]:

$$\nu_{lb} = \frac{1}{3} \left(\tau - \frac{1}{2} \right) \tag{2.19}$$

This factor takes a very important role in the numerical stability of the method.

When (2.18) is introduced in (2.15) the Boltzmann equation with the BGK approximation results

$$\frac{\partial f}{\partial t} + \vec{v} \nabla f = \frac{1}{\tau} \left(f^{eq} - f \right) \tag{2.20}$$

2.4 The Discrete Boltzmann Equation.

Equation (2.20) is solved by the LBM every time step to simulate the fluids. Moreover, this equation works on a more fundamental level than NS equations, so it is possible to derive NS equations from the Boltzmann equation

[Moh11]. Equation (2.20) can be discretized [SJ06][Moh11] as

$$f_i\,(\vec{r}\,+\,\vec{e_i}\Delta t,\,t\,+\,\Delta t)\;=\;f_i\,(\vec{r}\,,t\,)+\frac{\Delta t}{\tau}[f_i^{eq}(\vec{r},\,t)\,-\,f_i(\vec{r},\,t)] \tag{2.21}$$

In this discrete form of the *Boltzmann equation* it can be seen the two basic steps in the LBM.

$$\underbrace{f_i\,(\vec{r}\,+\,\vec{e_i}\Delta t,\,t\,+\,\Delta t)\;=\;f_i\,(\vec{r}\,,t\,)}_{\text{streaming step}}+\underbrace{\frac{\Delta t}{\tau}[f_i^{eq}(\vec{r},\,t)\,-\,f_i(\vec{r},\,t)]}_{\text{collision step}} \tag{2.22}$$

The beauty of this equation is its simplicity. Thus, it can be easily modified to introduce different simulation properties.

2.5 The Lattice

In the LBM the discrete Boltzmann Equation is solved locally in every node of a fixed grid or lattice to update the values of the different variable fields and simulate the fluid behaviour. Each lattice node can be related to its neighbouring nodes in the streaming step through the fixed directions that link every node with its neighbouring nodes (see Fig. 2.2). This fixed number of links between the node and its neighbours is the consequence of the discretization of the continuum velocity field to a discrete number of directions and magnitudes. The number of directions and its magnitudes depend on every lattice arrangement. In addition, on each lattice node resides the distribution function.

Hence, this can be understood as if on each lattice node there is a group of molecules, where subgroups of this first group have a velocity term in certain direction (the portion of molecules in these subroups is calculed through the distribution function). It can be seen that the distance between the node and its neighbour are not always the same. For instance, the distance between one node and its north-west neighbour is larger than the distance between

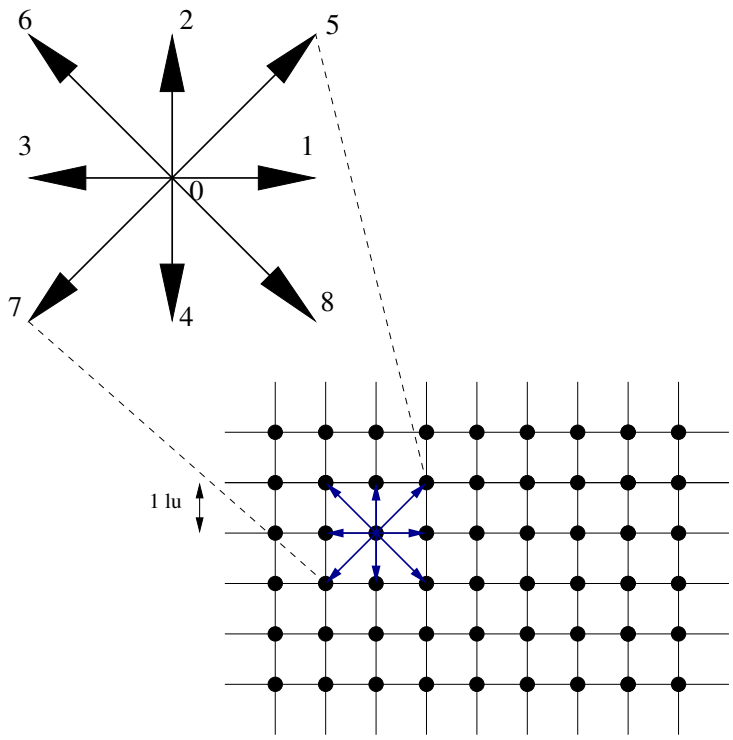


Figure 2.2: 2D lattice example.

the same node and its west neighbour. This difference has to be taken into account when calculating the equilibrium distribution function values for the collision step (see section 2.9).

2.6 Lattice configurations.

There are different sort of lattices configurations and they are identified through their name. The nomenclature used in the literature to refer the lattice different possible arrangements is the following: $DdQq$ where d says in how many dimensions the lattice model is set, and q how many possible directions can have the velocity. For example D2Q9, where the lattice is spread in two dimensions (2D) and it has 9 possible velocity directions, including the velocity zero for non-moving particles that stay a the current node location. In this thesis two different types of lattices arrangements have

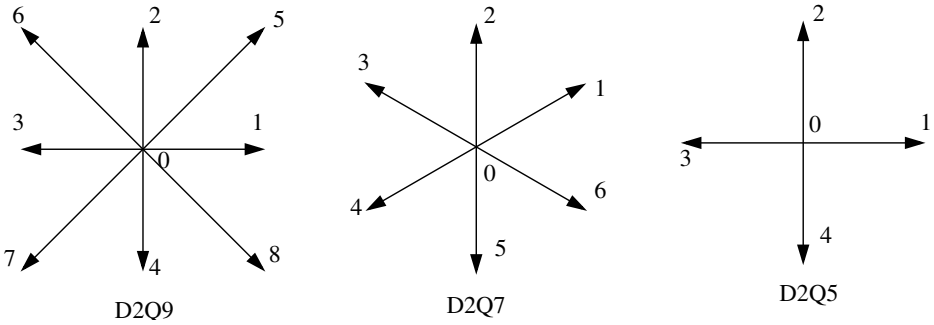


Figure 2.3: D2Q9, D2Q7 and D2Q5 arrangements

been used. In the early stages of the C++ code development, the D2Q9 arrangement has been used for its simplicity and to validate the method using simple geometries and boundary conditions. It is also used to describe some concepts due to its simplicity.

As it can be seen in Fig. 2.3 the lattice D2Q9 configuration can have 9 different velocity directions (8 to the periphery and 1 in the centre), with the following vectors

$$\vec{e_0} = (0,0) \qquad \vec{e_1} = (1,0) \qquad \vec{e_2} = (0,1)$$
$$\vec{e_3} = (-1,0) \qquad \vec{e_4} = (0,-1) \qquad \vec{e_5} = (1,1)$$
$$\vec{e_6} = (-1,1) \qquad \vec{e_7} = (-1,-1) \qquad \vec{e_8} = (1,-1)$$

From the mathematical point of view, these vectors represent two different considerations that must be separed. The first one is that the velocity vectors represent elements of a translation space, thereby they are coordinate-independent. The second consideration about the velocity vectors is that they represent a local spatial basis, therefore, they are not coordinate-independent. The first ones have indeed only a length (not a distance) while the latter ones represent a distance.
It can be seen that in the discrete domain of the lattice the distance between the vector and its neighbour is not always the same, *i.e.* $\vec{e_1}$ has a length of

one lattice unit (lu) meanwhile $\vec{e_5}$ has a length of $\sqrt{2}\,lu$. This has its effect on the treatment of the data, because in the discrete domain the discrete $dt = \Delta t$ is constant, so it means that the velocity magnitude is different as well, being this magnitude equal to 1 from $\vec{e_1}$ to $\vec{e_4}$, and $\sqrt{2}$ from $\vec{e_5}$ to $\vec{e_8}$. The velocity magnitude of $\vec{e_0}$ is null.

The weights factors for the D2Q9 arrangement, which are derived in [Sat10], are these ones

$$w_0 = \frac{4}{9} \qquad w_{i=1\ldots4} = \frac{1}{9} \qquad w_{i=5\ldots8} = \frac{1}{36}$$

In the 3D domain there are also different options of configurations. The one used in this thesis is the D3Q19 which is one of the most typical three-dimensional lattice types. It has been shown that the increase of the number of possible velocity directions increases the accuracy of the model [AEM09], but it also increases the computational effort. Hence, D3Q19 has a better accuracy compared to the D3Q15, but the increase in the spent time is sizable. On the other hand, it is shown that the accuracy of D3Q27 model is indeed higher than the D3Q19 system, but it requires a much higher computational effort, which is not compensated by the corresponding accuracy gain.

In Fig. 2.4 it can be seen the difference between this three-dimensional lattice configurations. In the D3Q19 arrangement the corresponding velocity directions are

$$\begin{array}{llll}
\vec{e_0} = (1,0,0) & \vec{e_1} = (-1,0,0) & \vec{e_2} = (0,1,0) & \vec{e_3} = (0,-1,0) \\
\vec{e_4} = (0,0,1) & \vec{e_5} = (0,0,-1) & \vec{e_6} = (1,1,0) & \vec{e_7} = (1,-1,0) \\
\vec{e_8} = (1,0,1) & \vec{e_9} = (1,0,-1) & \vec{e_{10}} = (-1,1,0) & \vec{e_{11}} = (-1,-1,0) \\
\vec{e_{12}} = (-1,0,1) & \vec{e_{13}} = (-1,0,-1) & \vec{e_{14}} = (0,1,1) & \vec{e_{15}} = (0,1,-1) \\
\vec{e_{16}} = (0,-1,1) & \vec{e_{17}} = (0,-1,-1) & \vec{e_{18}} = (0,0,0) &
\end{array}$$

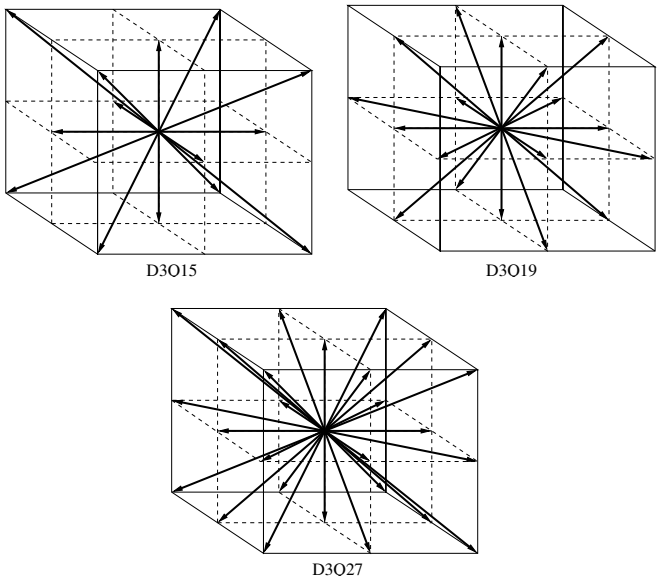


Figure 2.4: 3D lattice models

and the corresponding weights are [Sat10]

$$w_{i=0...5} = \frac{1}{18} \qquad w_{i=6...17} = \frac{1}{36} \qquad w_{18} = \frac{1}{3}$$

In Fig. 2.5 it can be seen the definition of the velocity direction vectors in the D3Q19 lattice.

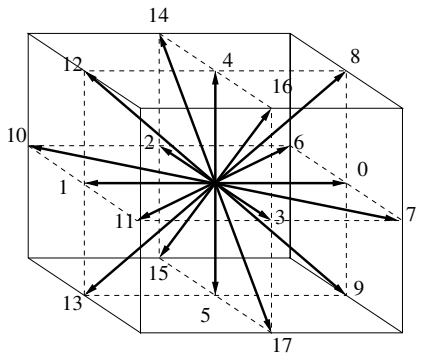


Figure 2.5: D3Q19 lattice

2.7 Macroscopic Quantities

The *Boltzmann Equation* works in a microscopic scale and only the macroscopic variables are experimentally accessible and relevant for engineering design of macroscopic thermohydraulic devices. Thus, only via these macroscopic variables, simulations or even theoretically obtained results can be compared to the experimental data.

The macroscopic variables can be calculated in a straight manner from the values of the distribution function [Thu03]. The density,

$$\rho\,(\vec{r}\,,t) \;=\; \int m\, f(\vec{r}\,,\vec{v}\,,t) d^3v \tag{2.23}$$

and the macroscopic velocity $\vec{u}$,

$$\rho\,(\vec{r}\,,t)\,\vec{u}(\vec{r},\,t) \;=\; \int m\,\vec{v}\, f(\vec{r}\,,\vec{v}\,,t) d^3v \tag{2.24}$$

Thus, in the discrete domain the calculus of these macroscopic variables is as well straightforward [Thu03][SJ06]

$$\rho = \sum_{a=0}^{q} f_a \tag{2.25}$$

$$\vec{u} = \frac{1}{\rho} \sum_{a=0}^{q} f_a \vec{e_a} \tag{2.26}$$

where q stands for the number of discrete velocities that the chosen lattice model chosen has, and the mass m had been chosen to be unity. The discrete macroscopic velocity is an average value of the microscopic velocities $\vec{e_a}$ weighted by the directional densities f_a.

The discrete macroscopic pressure is given by the EOS that relates the discrete density to the pressure by a simple proportional relation taking into

account the speed of sound [SJ06]

$$P \;=\; c_S^2 \rho \;=\; \frac{\rho}{3} \tag{2.27}$$

Although this relation is valid for the method, one has to take into account that this model is valid for simulations of incompressible fluids. This means that the density of the fluid is not allowed to vary, and it is only allowed to fluctuate locally around a fixed value. This restriction limits the method and, thus, it is not valid to simulate situations where compressibility plays an important role, such as the shock wave simulations or even sound waves.

2.8 Streaming and Collision step

The streaming step follows this equation

$$f_i\,(\vec{r} \,+\, \vec{e_i}\Delta t,\, t \,+\, \Delta t) \;=\; f_i\,(\vec{r}, t\,) \tag{2.28}$$

and it is the transfer of the direction-specific densities f_i to the nearest neighbour lattice nodes. This can be understood as if a group of the molecules located on the node moved to the nearest neighbour modifying its density and thus its velocity. During the numerical implementation of the streaming step one has to take care not to overwrite the distribution function values during the operation.
The collision step is the relaxation of the new distribution function values, actualized by the streaming step, towards the equilibrium distribution.

$$\Omega \;=\; \frac{\Delta t}{\tau}[f_i^{eq}(\vec{r},\, t) \;-\; f_i(\vec{r},\, t)] \tag{2.29}$$

This collision step follows Eq. (2.29) and introduces the effect of the collisions between particles in the model.

In Fig. 2.6 can be seen the effect of the application of the two steps in

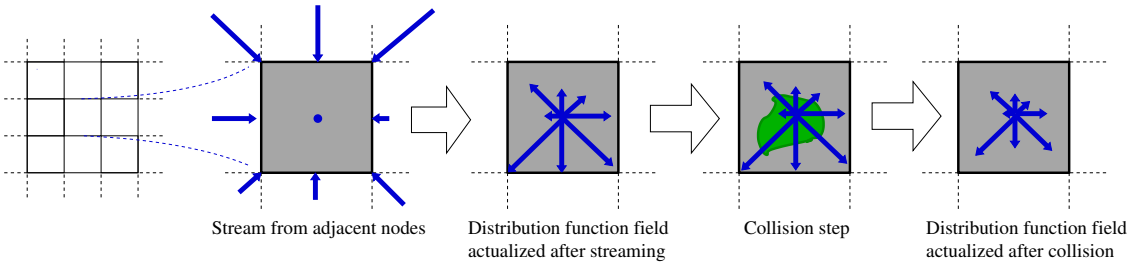


Figure 2.6: Streaming and Collision step.

a specific node. The arrows length represent the normalized values of the distribution function for every discrete direction. It is important to notice that the length of the arrows is not homogeneous (as seen in previos schematic figures used to explain the lattice configurations). If the particle had null velocity (*i.e.* the distribution function values are the equilibrium distribution function values) the arrows length would be homogeneous. In this example the node has a given velocity, therefore the distribution function values differ from the equilibrium distribution function.

2.9 Equilibrium distribution function

The equilibrium distribution function f^{eq} that appears in the collision operator is crucial element in the LBM. This distribution function is the *Maxwell-Boltzmann distribution*, which is expanded for low *Mach* number [Moh11] to ensure that the fluid can be considered incompressible. This is a computational simplification.

It is known that a flow can neglect the compressibility effects when it has a low *Mach* number (typically $Ma < 0.3$). The normalized *Maxwell-Boltzmann* distribution is [Mel13][HL97]:

$$f_a = \frac{\rho}{2\pi/3} e^{-\frac{3}{2}(\vec{e_a}-\vec{u})^2} = \frac{\rho}{2\pi/3} e^{-\frac{3}{2}(\vec{e_a}\cdot\vec{e_a})} e^{\frac{3}{2}(2\vec{e_a}\vec{u}-\vec{u}\cdot\vec{u})} \tag{2.30}$$

where $\vec{u}$ is the macroscopic velocity and $\vec{e}$ are the velocity vectors in the lattice configuration. As it can be seen in Eq. (2.30) the macroscopic velocity $\vec{u}$ is compared to the lattice speed $\vec{e}$. Thereby, since the speed of sound is the reference in the lattice framework, if $\vec{u} = 1$ it could be taken as if the lattice speed was the speed of sound. Thus, the velocity in the lattice must be kept much smaller than the lattice speed of sound to ensure incompressibility. If the lattice speed exceeds the lattice speed of sound ($\vec{u} > 1$), it would mean that the information could not be transpoted from one lattice node to another. It can be understood as a shock wave limit.

Now if Eq. (2.30) is expanded using the *Taylor* polynomial expansion for exponentials, Eq. (2.31) to the second order to match with the NS equations order.

$$e^x \;=\; 1 \,+\, x \,+\, \frac{x}{2!} \,+\, \frac{x}{3!} \,+\, ... \tag{2.31}$$

$$f \;=\; \frac{\rho}{2\pi/3} e^{-\frac{3}{2}(\vec{e_a}\cdot\vec{e_a})}\big[1 \,+\, 3(\vec{e_a}\cdot\vec{u}) \,-\, \frac{3}{2}(\vec{u}\cdot\vec{u}) \,+\, \frac{9}{2}(\vec{e_a}\cdot\vec{u})^2\big] \tag{2.32}$$

Thus, the equilibrium distribution function takes the following form [Moh11]:

$$f_a^{eq} \;=\; \rho\,\omega_a\big[1 \,+\, 3(\vec{e_a}\cdot\vec{u}) \,-\, \frac{3}{2}(\vec{u}\cdot\vec{u}) \,+\, \frac{9}{2}(\vec{e_a}\cdot\vec{u})^2\big] \tag{2.33}$$

In the implementation in `C++` of the computation of the equilibrium distribution function values, one has to take care that there are dot products between the vectors involved.

2.10 Lattice Boltzmann Method Algorithm

The LBM algorithm consists essentially of the two basic steps explained in section 2.8, the streaming step and the collision step. But there are also different steps that must be applied sequentially. Below, the schematic sequence of the algorithm is shown.

```
Initialize;
for number of iterations do
    Apply boundary conditions;
    Streaming Step;
    Compute macroscopic quantities;
    Compute f^eq;
    Collision Step;
    if there are obstacles then
        Obstacle treatment;
    end
end
Store data;
```

Algorithm 1: Main algorithm of the LBM

It is true that the LBM main algorithm can be implemented in different manners, but the initialization step must strictly be at the beginning. Then, the streaming and collision stem must be done in a strictly alternating sequence. In algorithm 1 there is the sequence that has been implemented during the development of this thesis.

2.10.1 Initialization

The initialization procedure consists in several steps. The first one is to read the input data introduced by the user and allocate the memory and variables, this is purely and informatic step. The second step is to initialize the dimensionless problem from the dimensioned characteristic values entered by the user. The third basic step is to initialize the phase space distribution from a macroscopic velocity and density field. For instance, a common phase space distribution is the equilibrium distribution where the macroscopic velocity is null.

It can be seen the schematic sequence of the initialization subroutine in algorithm 2

Result: Initialization of the algorithm
Get simulation parameters;
Create lattice;
Read geometry file;
Define node properties in function of geometry;
Classify nodes in lists;
Initialize velocity;
Compute f^{eq};
Initialize f;

Algorithm 2: Initialization routine

To see it more clearly, this process can be understood as follows: the input data defines the simulation parameters (*i.e.* Reynolds number, lattice dimensions, initial velocities...). Then, after a classification of nodes, done purely due to the implementation structure needs, the initial value of the velocity is placed in every node. For example, if the fluid is in an equilibrium state at the initial time the value of initial velocity would be zero and then this value would be stored in the nodal value for velocity. Afterwards, when the equilibrium distribution function would be calculated, it would use for this calculus the nodal value for velocity stored before.
A common initial value for the distribution function is the equilibrium distribution value calculated previously taking into account the initial value of the velocity.

$$f_{initial} = f^{eq}(u_{initial})$$

In the appendix A there will be a further explanation of how the algorithm is initialized taking into account the structure and hierarchy chosen to implement the algorithm.

2.10.2 Iteration loop

In the iteration loop there is some flexibility on when to apply some steps or the others. The following order is the one that has been followed in the implementation of the algorithm.

Apply boundary conditions;
Streaming Step;
Compute macroscopic quantities;
Compute f^{eq};
Collision Step;
if *there are obstacles* **then**
  Obstacle treatment;
end

Algorithm 3: Iteration loop of the LBM

Here is a further explanation of some steps.

Boundary conditions

The boundary conditions are indeed a key part of the method. They will be discussed further in subsection 2.11, but one has to know that they are applied in the main loop systematically. It means that every time step the boundary conditions are applied to the boundary nodes to actualize and modify the distribution function values in certain directions

Calculus of the macroscopic variables (ρ, u) and equilibrium distribution function (f^{eq}).

In every time step the algorithm has to update the macroscopic variables for the calculus of the equilibrium distribution function needed in the collision step.

2.11 Boundary Conditions

The boundary conditions in the LBM are not intuitive. Usually one has macroscopic boundary condition (*e.g.* constant velocity or constant pressure) which does either not uniquely defines the microscopic boundary conditions,

or even does not translate to microscopic boundary conditions (*e.g.* the pressure boundaries) due to the mismatch of macroscopic approximation.

There are several types of boundary conditions, and in this chapter there are the definition of those which have been used in this thesis for the two different lattice arrangements that have been used, the D2Q9 and D3Q19.

2.11.1 Bounce-back boundaries

These boundaries, also called *no-slip condition* boundaries, are the typical boundaries for simulating the interaction of fluids with a non-moving wall without slip. Also, they are used to simulate the flow around a stationary obstacle. They are largely used because they ensure the mass, momentum and energy conservation [Moh11], and due to their simplicity, their numerical stability and their accuracy.
As the name implies, when a particle is coming towards the solid boundary it bounces back into the flow domain [SJ06][CFHL09]. This can be seen in Fig. 2.7

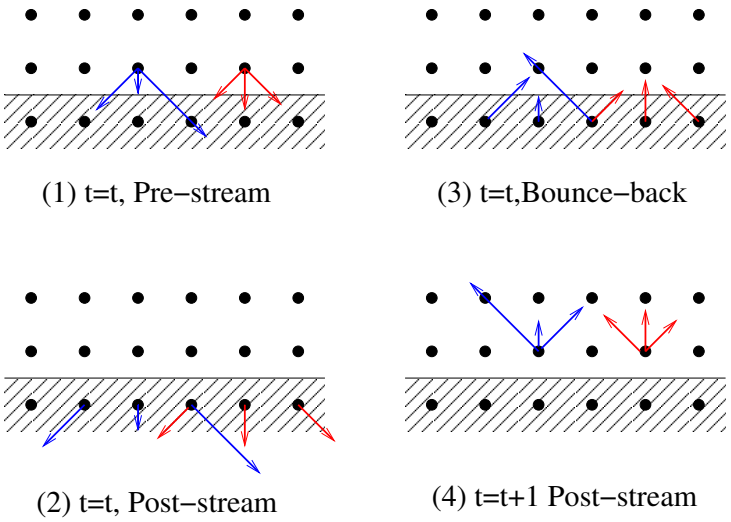


Figure 2.7: Bounce-back boundary condition effect on the distribution function values.

There are two basics schemes to implement the bounce-back boundaries, and they have different results. The first one suggests that the solid wall should be placed half way between the boundary node and the fluid node. The other scheme suggests placing the solid on a node directly. It has been shown that the first scheme is second-order in numerical error, and the second scheme is first-order in numerical error[Mel13][GNGB97].

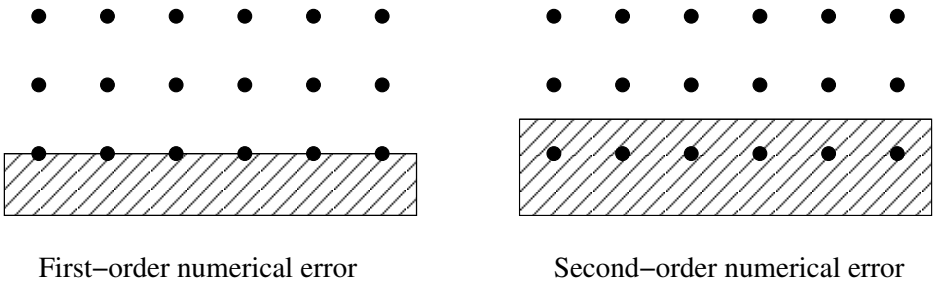


Figure 2.8: The two different approaches for bounce-back boundaries

Therefore, as shown in Fig. 2.8, one has to take this into account when the geometry is defined. Accuracy can always be increased by increasing the lattice resolution, but in a three-dimensional lattice the memory needed to store all the data is rather large, so it implies a large computational cost that can be avoid only by taking care when defining the geometry and lattice.

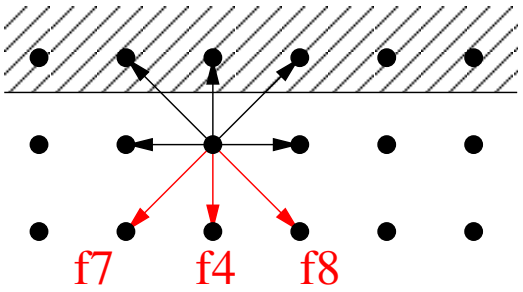


Figure 2.9: D2Q9 Bounce-back boundary in the north direction

The explanation of the equations that one has to use to apply the bounce-

back boundary condition will be focused on the top boundary as shown in Fig. 2.9. If the node in contact with a boundary node is considered, it can be seen that there is no streamed distribution function values from the solid node. Therefore, there are three unknowns in the distribution function field of the fluid node, f_4, f_7 and f_8. This values can be obtain by this relations

$$f_4 = f_2 \qquad f_7 = f_5 \qquad f_8 = f_6$$

With these relations it can be seen that, since the streaming does not provide any value for f_4, f_7 and f_8, it is the bounce-back conditions which provides the values for this unknowns.

The derivation of the bounce-back relations for the D3Q19 from the D2Q9 relations is straightforward [HH10a] and the only difference between the D2Q9 model and the D3Q19 is the amount of unknowns that have to be solved. The equations that have to be applied in the three-dimensional bounce-back boundaries, for the north boundary, are the following ones

$$f_5 = f_4 \qquad f_{13} = f_8 \qquad f_9 = f_{12}$$
$$f_{15} = f_{16} \qquad f_{17} = f_{14}$$

2.11.2 Periodic boundaries

Periodic boundary conditions are useful in some cases. Although they do not simulate reality they can be used in some benchmarks to validate the system, for example.

When the periodic boundaries are applied in one direction the domain of the simulation changes to cylindrical geometry as shown in Fig. 2.10.

The nodes placed in the boundary, where the periodic condition is applied, have its neighbouring nodes on the opposite boundary. The explanation of

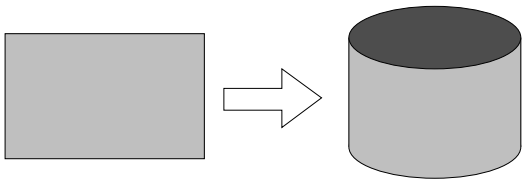


Figure 2.10: Cylindrical shape of the domain when periodic boundary conditions are applied.

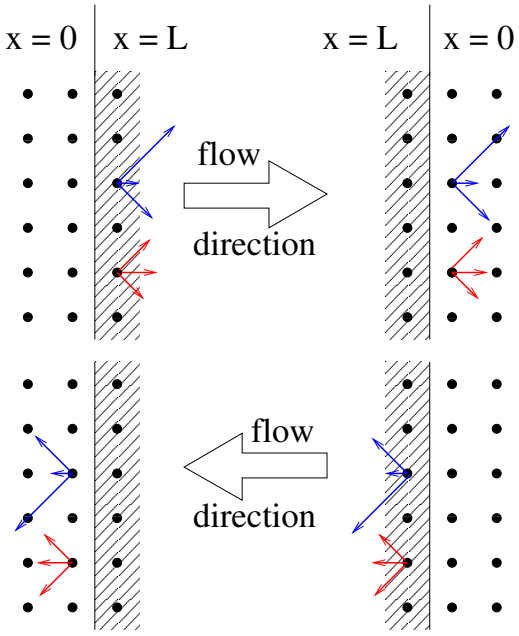


Figure 2.11: Effect of the periodic boundary condition on the boundary node.

the periodic boundaries will be focused on the x-direction case, explained in Fig. 2.11.

Therefore in the D2Q9 model the relations that needed to be applied are the following

$$f_1^{x=0} = f_1^{x=L} \qquad f_5^{x=0} = f_5^{x=L} \qquad f_8^{x=0} = f_5^{x=L}$$
$$f_3^{x=L} = f_3^{x=0} \qquad f_6^{x=L} = f_6^{x=0} \qquad f_7^{x=L} = f_7^{x=0}$$

the relations for the D3Q19 model can be extrapolated from the D2Q9 model as follows

$$
\begin{aligned}
f_0^{x=0} &= f_0^{x=L} & \qquad f_1^{x=L} &= f_1^{x=0} \\
f_6^{x=0} &= f_6^{x=L} & \qquad f_{10}^{x=L} &= f_{10}^{x=0} \\
f_7^{x=0} &= f_7^{x=L} & \qquad f_{11}^{x=L} &= f_{11}^{x=0} \\
f_8^{x=0} &= f_8^{x=L} & \qquad f_{12}^{x=L} &= f_{12}^{x=0} \\
f_9^{x=0} &= f_9^{x=L} & \qquad f_{13}^{x=L} &= f_{13}^{x=0}
\end{aligned}
$$

2.11.3 Velocity boundaries (Von Neumann boundaries)

It is clear that there is no complex simulation that could be simulated properly only with periodic and bounce-back boundaries configuration. Therefore, *Zou and He* [ZH98] introduced in 1997 the velocity boundary conditions (also called Von Neumann boundary conditions) that apply a certain flux condition on the boundary.
In these conditions a velocity is specified $\vec{u} = (u_x, u_y, u_z)$ from which a density is computed, and thus, the distribution function values are calculated to achieve the distribution that implies the fixed velocity.
The derivation of these boundary conditions can be seen also in [SJ06], [CFHL09] and [ZH98]. Here, the explanation of the process to derive the conditions that must be applied, is restricted for the top node in the D2Q9 configuration (see Fig 2.9).

Along the boundary there are also unknown distribution function values that needed to be solved. These distribution function values can be expressed as a combination of the local known value and a corrector term [CFHL09]

$$f_a(\vec{r}, t) = f_a^*(\vec{r}, t) + \omega_a \vec{e_a} \vec{Q} \tag{2.34}$$

where term $\vec{Q}$ is the force corrector to apply the required momentum, and

f_a^* is the opposite distribution function value for f_a. If the top boundary is considered, f_4, f_7 and f_8 are the unknown density distribution functions, and they can be expressed by Eq(2.34) as follows.

$$\begin{aligned} f_4(\vec{r},t) &= f_4^*(\vec{r},t) + \omega_4 \vec{e_4}\vec{Q} \\ f_7(\vec{r},t) &= f_7^*(\vec{r},t) + \omega_7 \vec{e_7}\vec{Q} \\ f_8(\vec{r},t) &= f_8^*(\vec{r},t) + \omega_8 \vec{e_8}\vec{Q} \end{aligned} \tag{2.35}$$

Therefore, this relations can be included in equations Eq. (2.25) and Eq. (2.26) [CFHL09], and thus:

$$\begin{aligned} \rho &= f_0 + f_1 + f_2 + f_3 + (f_4^* - \omega_4 Q_y) + f_5 \\ &\quad + f_6 + (f_7^* - \omega_7(Q_x + Q_y)) + (f_8^* + \omega_8(Q_x - Q_y)) \\ \rho u_x &= f_1 + f_5 + (f_8^* + \omega_8(Q_x - Q_y)) - f_3 \\ &\quad - f_6 - (f_7^* - \omega_7(Q_x + Q_y)) \\ \rho u_y &= f_2 + f_5 + f_6 + (f_4^* - \omega_4 Q_y)) \\ &\quad - (f_7^* - \omega_7(Q_x + Q_y)) - (f_8^* + \omega_8(Q_x - Q_y)) \end{aligned} \tag{2.36}$$

where u_x and u_y are known because they are fixed at the beginning. Nevertheless there is still needed to solve f^* for every distribution function. There are three different approaches [CFHL09]:

1. $f_a^*(\vec{r},t) = f(\vec{r}, -\vec{e_a}, t)$
2. $f_a^*(\vec{r},t) = f(\vec{r}, \vec{e_a}, t - dt)$
3. $f_a^*(\vec{r},t) = f^{eq}(\vec{r}, \vec{e_a}, t)$

The first approach is the one that accomplishes and recovers the form firstly developed [ZH98] and it is the one adopted in this thesis. Then, equations (2.33) can be used to found ρ, Q_x and Q_y.

Thus, the explicit equations that are needed to be solved to apply the velocity boundary conditions in the top boundary are [CFHL09]

$$\begin{aligned}
\rho &= \frac{f_0 + f_1 + f_3 + 2(f_2 + f_5 + f_6)}{1 + u_y} \\
f_4 &= f_2 - \frac{2}{3}\rho u_y \\
f_7 &= f_5 - \frac{1}{2}\rho u_x - \frac{1}{6}\rho u_y + \frac{1}{2}(f_1 - f_3) \\
f_8 &= f_6 - \frac{1}{2}\rho u_x - \frac{1}{6}\rho u_y - \frac{1}{2}(f_1 - f_3)
\end{aligned} \tag{2.37}$$

If the same procedure is applied to the D3Q19 model [CFHL09][HH10a], the equations to apply at the top boundary are

$$\begin{aligned}
\rho &= \frac{1}{u_z + 1}[f_0 + f_1 + f_2 + f_3 + f_6 + f_7 + f_{10} + f_{11} + f_{18} \\
&\quad + 2(f_4 + f_8 + f_{12} + f_{14} + f_{16})] \\
f_5 &= f_4 - \frac{1}{3}\rho u_z \\
f_9 &= f_{12} + \frac{\rho}{6}(-u_z + u_x) - Q_x^z \\
f_{13} &= f_8 + \frac{\rho}{6}(-u_z - u_x) + Q_x^z \\
f_{15} &= f_{16} + \frac{\rho}{6}(-u_z + u_y) - Q_y^z \\
f_{17} &= f_{14} + \frac{\rho}{6}(-u_z - u_y) + Q_y^z \\
Q_x^z &= \frac{1}{2}[f_0 + f_6 + f_7 - (f_1 + f_{10} + f_{11})] - \frac{1}{3}\rho u_x \\
Q_y^z &= \frac{1}{2}[f_2 + f_6 + f_{10} - (f_3 + f_7 + f_{11})] - \frac{1}{3}\rho u_y
\end{aligned} \tag{2.38}$$

2.11.4 Pressure boundaries (Dirichlet boundaries)

In 1997 together with the velocity boundaries, *Zou and He* [ZH98] introduced as well the pressure boundaries. These boundaries, analogously to the velocity boundaries, calculate the unknown distribution function values using the imposed density.

It is known that the LBM used in this thesis is for incompressible fluids. Therefore, as the fluid is incompressible, its density is fixed. From the EOS that relates the pressure and density (2.27), one can think that the pressure must be constant also, but in reality the density is allowed to fluctuate around a constant fixed value, to allow the fluid to have pressure gradients, and therefore, velocity. Nonetheless, the EOS is an equation that is needed to enclose the system, and, therefore, it is an approximation.
The pressure is fixed using this EOS, and thus, one can use the derivation made in the Von Neumann boundaries and use it to establish the equations that have to be solved for a top pressure boundary.
For the D2Q9 model, and the top boundary, the equations are

$$\begin{aligned} u_y &= -1 + \frac{1}{\rho}[f_0 + f_1 + f_3 + 2(f_2 + f_5 + f_6)] \\ f_4 &= f_2 - \frac{2}{3}\rho u_y \\ f_7 &= f_5 - \frac{1}{2}\rho u_x - \frac{1}{6}\rho u_y + \frac{1}{2}(f_1 - f_3) \\ f_8 &= f_6 - \frac{1}{2}\rho u_x - \frac{1}{6}\rho u_y - \frac{1}{2}(f_1 - f_3) \end{aligned} \tag{2.39}$$

If the same procedure is applied to the D3Q19 model also for the top boundary

$$
\begin{aligned}
u_z &= -1 + \frac{1}{\rho}[f_0 + f_1 + f_2 + f_3 + f_6 + f_7 + f_{10} + f_{11} + f_{18} \\
&\quad + 2(f_4 + f_8 + f_{12} + f_{14} + f_{16})] \\
f_5 &= f_4 - \frac{1}{3}\rho u_z \\
f_9 &= f_{12} + \frac{\rho}{6}(-u_z + u_x) - Q_x^z \\
f_{13} &= f_8 + \frac{\rho}{6}(-u_z - u_x) + Q_x^z \\
f_{15} &= f_{16} + \frac{\rho}{6}(-u_z + u_y) - Q_y^z \\
f_{17} &= f_{14} + \frac{\rho}{6}(-u_z - u_y) + Q_y^z \\
Q_x^z &= \frac{1}{2}[f_0 + f_6 + f_7 - (f_1 + f_{10} + f_{11})] - \frac{1}{3}\rho u_x \\
Q_y^z &= \frac{1}{2}[f_2 + f_6 + f_{10} - (f_3 + f_7 + f_{11})] - \frac{1}{3}\rho u_y
\end{aligned}
\tag{2.40}
$$

2.11.5 Open boundary conditions

This kind of boundaries is applied when the outlet velocity in the system outlet is not known. Typically an extrapolation can be made [Moh11], and the distribution function value needed can be calculated from the adjacent nodes as shown in Fig 2.12. The equations shown here correspond to the right boundary

$$
\begin{aligned}
f_{3,n} &= 2f_{3,n-1} - f_{3,n-2} \\
f_{6,n} &= 2f_{6,n-1} - f_{6,n-2} \\
f_{7,n} &= 2f_{7,n-1} - f_{7,n-2}
\end{aligned}
$$

There are different approaches when it comes to open boundaries [IMLF09], but in this thesis they are restricted to the ones explained above.

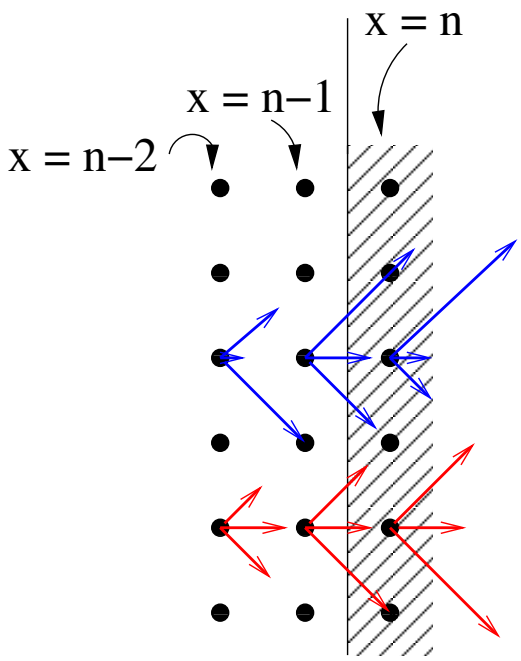


Figure 2.12: Open boundary condition scheme.

2.12 Physical and discrete domains. Dimensionalization issues.

In fluid dynamics simulations, the equations to solve underlie invariance under scaling of the physical variables. So, one only needs to solve one specific non-dimensional problem and scale to any size as long as the system is similar to the adimensional system.
There are two main domains involved in the LBM. The physical and the discrete. The physical domain is the one that defines the problem. There are several approaches on how to scale the physical parameters in to the lattice framework. The one followed here [Lat08] contains first an adimensionalization and then, a discretization of the space and time in the lattice framework. Thereby, velocity is also discrete in the lattice domain.

$$Physical\ system\ (P) \Leftrightarrow Dimensionless\ system\ (D) \Leftrightarrow Discrete\ system\ (LB)$$

Another approach is to pass directly from (P) to (LB), which they are equivalent by scaling, but one has to take care that the continuous consideration is not true in the discrete domain. In addition, there are several reasons why it is better to take an intermediate step [Lat08]. The first reason is the choice of Δx and Δt variables which are important parameters that have an impact on the accuracy and numerical stability of the simulation.
Using the Reynolds number, which is a dimensionless number, one can relate the dimensional and dimensionless variables. Fixing a Reynolds number one can define a class of similar problems. Reynolds number also determines the flow regime [CK04] , low Re number for a laminar flow and high Re number for a turbulent flow.
The Reynolds number is defined as [CK04]

$$Re = \frac{u\,l}{\nu} \tag{2.41}$$

where u is the velocity in m/s, length l in m and ν the kinematic viscosity in m^2/s.

2.12.1 Dimensionless formulation.

To be able to make the dimensionless formulation first of all one has to define the physical problem with characteristic parameters.
When the characteristic values of the system are chosen, the parameters that need to pass from the physical to the dimensionless framework are

$$Re_{d,0} = \frac{u_0\,l_0}{\nu} \qquad\qquad u_p = \frac{l_{0,p}}{t_{0,p}} u_d$$

$$t_d = \frac{t_p}{t_{0,p}} \qquad\qquad r_d = r_p l_{0,p}$$

where the subindices $(\ )_d$ stands for dimensionless value, $(\ )_p$ for physical value, and $(\ )_{,0}$ stands for characteristic value either in the physical or dimensionless domain. One can see that if the Re is the same in (P) then in

(D) the problems can be considered similar.

$$Re_{p,0} = Re_{d,0}$$

The characteristic variables in the dimensionless framework are of the order of one [Lat08].

2.12.2 Discretization.

In the discrete framework $\Delta x_0 = \Delta t_0 = 1$. This is but just a scale. It is made this way that Δx_0 corresponds to the characteristic length. The time step $\Delta t_0 = 1$ is, also due to the chosen time scale, an iteration counter. Therefore the lattice speed of sound is also 1. After these parameters have been fixed, the lattice has two degrees of freedom, the discrete velocity u_{lb} and the viscosity in the lattice ν_{lb}. Therefore, the number of nodes inside the geometry N_x is fixed because the Reynolds number must be the same in the physical and in the discrete domain to be able to have the same simulation conditions.

$$\Delta x_0 = 1 \qquad\qquad \Delta t_0 = 1 \tag{2.42}$$

the relations between the variables in (D) and (LB) are as follows,

$$u_d = \frac{\Delta x}{\Delta t} u_{lb} \qquad\qquad \nu_d = \frac{1}{Re} = \frac{(\Delta x)^2}{\Delta t} \nu_{lb} \tag{2.43}$$

therefore,

$$u_{lb} = \frac{\Delta t}{\Delta x} u_d \qquad\qquad \nu_{lb} = \frac{1}{Re} \frac{\Delta t}{(\Delta x)^2} \tag{2.44}$$

One can easily see that when $\Delta x = \Delta x_0$, and $\Delta t = \Delta t_0$, that is, the discrete length and the discrete time are equal to selves characteristic values, the discrete velocity u_d is exactly the lattice speed of sound.

Indeed the discretization is straightforward, but one has to be careful. The discrete lattice velocity is set to one, therefore no speed front can move faster than this velocity. The following relation (2.36) must be true to ensure the incompressibility of the fluid.

$$\frac{\Delta x}{\Delta t c_s} = \frac{u_{lb}}{c_s} < 0.3 \rightarrow \Delta t < \frac{\Delta x}{\sqrt{3}} \tag{2.45}$$

What also can be done is fix parameters that had a relation between them. One has to verify first that the discretization is not inconsistent. For example

$$u_{lb} = 0.1 = \frac{\Delta t}{\Delta x} u_d \qquad\qquad \nu_{lb} = 0.03 = \frac{\Delta t}{(\Delta x)^2} \frac{1}{Re}$$

If $u_d = 1$,

$$\Delta x = 10 \Delta t \qquad\qquad 0.03 = \frac{\Delta t}{10^2 (\Delta t)^2} \frac{1}{Re}$$

therefore,

$$\Delta t = \frac{1}{3Re} \qquad\qquad \Delta x = \frac{10}{3Re}$$

Here it can be seen that for rather large values of the Reynolds number the Δt and Δx are very small, and thus, it can create numerical stability problems. One can observe that the incompressibility restriction (2.45) can be verified.

$$\Delta t < \frac{\Delta x}{\sqrt{3}} \qquad\qquad \sqrt{3} < 10$$

The critical relation is indeed the one that imposes the velocity value. If

this value is kept below $0.2 \sim 0.3$ the incompressibility restriction will be accomplished.

2.13 Introduction of the body force

There are several approaches to introduce an external force in the LBM model [BG00]

2.13.1 Introducing a term in f^{eq}

In the NS equations when a body force is introduced the term included is expressed in terms of gravity $-\rho\nabla\Phi$ [BG00], where $\nabla\Phi$ is the gravitational potential. If the density variation induced by the body force is negligible (*i.e.* the fluid is still considered incompressible) the body force can be expressed with an altered pressure $p \to p + \rho g\, z$, being z the vertical height, positive counting against the direction of gravity.
Hence, it can be derived that if a term is introduced in the equilibrium distribution function equation (2.33), the body force will be introduced in the model [BG00]. Thus

$$f_a^{eq} = \rho\,\omega_a\left[1 + 3(\vec{e_a}\cdot\vec{u}) - \frac{3}{2}(\vec{u}\cdot\vec{u}) + \frac{9}{2}(\vec{e_a}\cdot\vec{u})^2\right] + \frac{\Phi d}{q c_s^2} \tag{2.46}$$

where d is the considered spatial dimension and q is the number of discrete velocity directions of the lattice model (*i.e.* $q = 9$ in D2Q9 and $q = 19$ in D3Q19). c_s is the speed of sound in the lattice.

2.13.2 Calculating f^{eq} with an altered velocity

It has been shown in the past chapters that the value of f^{eq} in every node depends of its velocity. Therefore one can introduce the body force in the system by modifying the velocity used to calculate f^{eq} [Moh11][BG00]. Thus,

$$\rho\vec{u}^* = \rho\vec{u} + \tau\vec{F} \tag{2.47}$$

Where $\vec{u}$ is computed from equation (2.26). Therefore, to compute the f^{eq} distribution values it is used $\vec{u}^*$ instead of $\vec{u}$.

2.13.3 Adding an additional term to the *Boltzmann Equation*

Another method to introduce gravity to the system is to add a term in the collision operator that modifies the distribution function [BG00]. Thus, in the discrete *Boltzmann Equation* (2.21)

$$f_i\,(\vec{r}\,+\,\vec{e_i}\Delta t,\,t\,+\,\Delta t)\;=\;f_i\,(\vec{r}\,,t\,)+\frac{\Delta t}{\tau}[f_i^{eq}(\vec{r},\,t)\;-\;f_i(\vec{r},\,t)]$$

a term must be added in the collision operator [BG00]. Therefore,

$$\Omega(\vec{r},t)\;=\;-\frac{1}{\tau}[f_a(\vec{r},t)\;-\;f_a^{eq}(\vec{r},t)]\;+\;\frac{d}{q\,c_s^2}\vec{F}\vec{e_a} \tag{2.48}$$

where, as in 2.13.1, d is the dimension of the model and q is the number of possible discrete velocity direction of the lattice model. Also f^{eq} is computed from the equilibrium distribution function equation (2.33) unaltered.

In this approach the acceleration is shifted to the collision integral side, whereas, in the original Boltzmann equation, the acceleration happens through a drift term on the left hand side. But as mentioned at the beginning of this section, the problem is how to discritze this reasonably in a lattice framework with discrete and fixed velocities.

2.14 Numerical stability.

2.14.1 Relaxation parameter τ

The major problem concerning the numerical stability is the value of the τ parameter. This parameter is fixed during the initialization process (see

2.10.1) when the simulation parameters are determined. Therefore one has to be careful while discretizing the system because the τ value cannot change once fixed.
τ has an important role in the collision step where f_a is relaxed towards the f^{eq} value [Moh11][SJ06]. Therefore, to be able to know in which range should τ be fixed, one can make a stability analysis [Sch10].

Hence, fixing $f^{eq} = 0.85$ (arbitrary value) and having $f_{t=0} = 0.1$ as an initial value for the f value, the equation the following equation is used as a simplified model of LBM made to iterate and reach the equilibrium distribution

$$f_{t=T} = f_{t=T-1} + \frac{1}{\tau}[f_i^{eq}(\vec{r},\ (t = T-1)) \ -\ f_i(\vec{r},\ (t = T-1))] \qquad (2.49)$$

thus, for a range of $0.4 < \tau < 16$, the results areas shown in Figs. 2.13 and 2.14

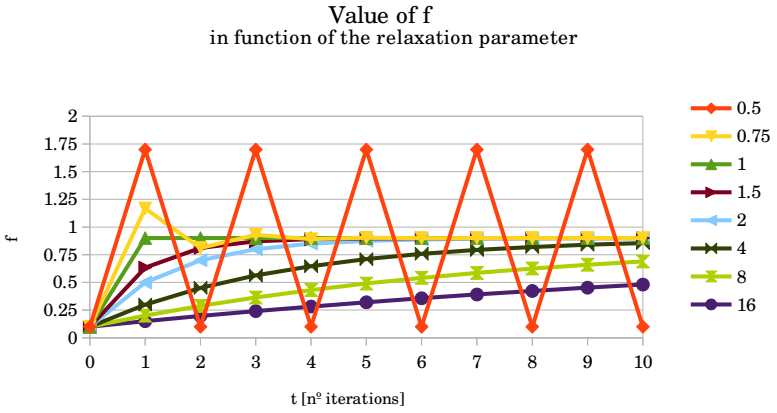


Figure 2.13: Evolution of f in function of the number of iterations with different values of τ

For values of $\tau > 0.5$ the asymptotic stability of the system is ensured. Nonetheless one has to be careful on how much time will it take to for f to reach the equilibrium. It is known by Eq. (2.19) that there is a relation between τ and the viscosity of the fluid ν, thus, the time to reach the equi-

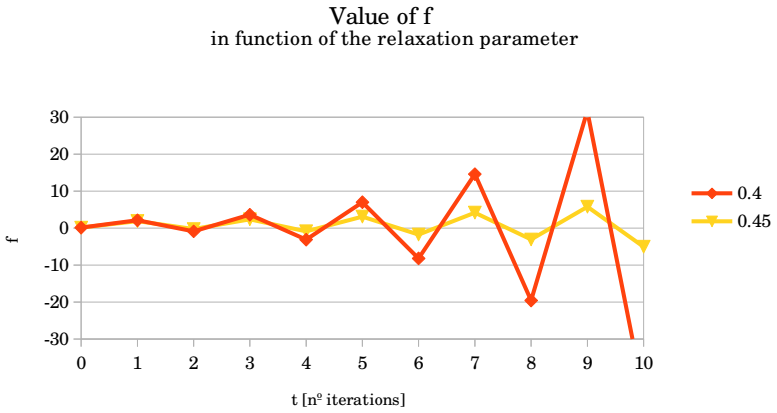


Figure 2.14: Instable evolution of f for values of $\tau < 0.5$

librium will be proportional to the viscosity as seen in Fig. 2.13.
Furthermore, if one takes a value of $\tau < 0.5$ as shown in Fig. 2.14, it can be seen that the system will never reach the stability. Therefore, a one must impose a limit for τ [SJ06] to ensure the numeric stability of the system. Theoretically, this limit has no upper value, but in practice τ should be limited to 5.

$$0.5 < \tau < \infty \longrightarrow 0.5 < \tau < 5$$

2.14.2 Discrete velocity u_{lb}

Another parameter that tends to create numerical instabilities in the method is the discrete velocity.
Discrete velocity must be kept below ~ 0.2 to maintain the incompressibility condition of the fluid. Whenever this condition is not ensured the whole model is not valid anymore.
Furthermore when u_{lb} has a value beyond this threshold the method becomes numerically instable and crashes.

2.15 Visualization tools

Supposing a lattice of rather large dimensions, the amount of data generated is so large. Hence, one needs extern software to be able to interpret the results. In addition to the office suites that have programs specifically designed to perform data analysis, there are two basic software tools used in this thesis, the *Gnuplot* [GNU] tool and the *OpenDX* [DX] data plotting tool.

2.15.1 Gnuplot

It is a multiplatform open source command-line graphing utility. It is the tool used to plot the *ASCII* format files generated by the algorithm (see A.1).

Gnuplot is a plotting tool used in 2-dimensional graphics. For instance, Fig. 2.15.2 is plotted with Gnuplot. It is a representation of the velocity field of a cut perpendicular to a Poiseuille flow in a squared pipe with bounce-back boundaries to simulate the pipe walls. This is only an example of the

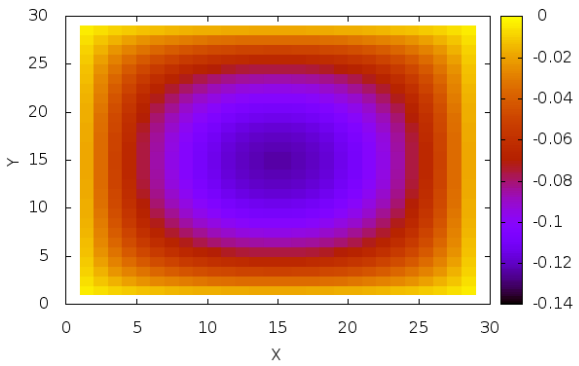


Figure 2.15: Gnuplot example

Gnuplot capabilities, the plots can be modified by different commands.
In chapters 3 and 4 there are more graphics plotted using Gnuplot.

2.15.2 OpenDX

OpenDX is a graphic tool more suited to plot 3D data fields. In contrast to Gnuplot, OpenDX is a tool that has a powerful user interface. This interface represents the plotting process using a sequential scheme of blocks where the data flow and is modified by these blocks. Every block has a specific effect on the data.
OpenDX allows the user to plot either scalar field or vector field. Of course, a 3D vector field in 3D is not the clearest way to plot it, but it might help to understand the behaviour of the algorithm.

Chapter 3

Validation of the LBM - Simulation description, results and analysis of the results

3.1 Poiseuille flow

The Poiseuille flow is the kind of flow that follows the *Hagen-Poiseuille-Law* [Wika]. This is a flow pattern of an incompressible, viscous flow with zero-boundary conditions. The Poiseuille flow solution can be derived from the NS equations [CK04].
Since it can be described analytically, its simulation is a typical benchmark to test numerical solution schemes. In this case several simulations have been done in order to test different boundaries configuration in the LBM scheme. The shape of the flow pattern can be seen in Fig. 3.1

In this study, velocity boundary conditions will be applied at the inlet and outlet surfaces of the domain. In addition, the whole fluid is enclosed in a pipe with circular cross-section where bounce-back boundary are applied. Therefore a velocity profile according to the Hagen-Poiseuille-law is developed inside the circular pipe.

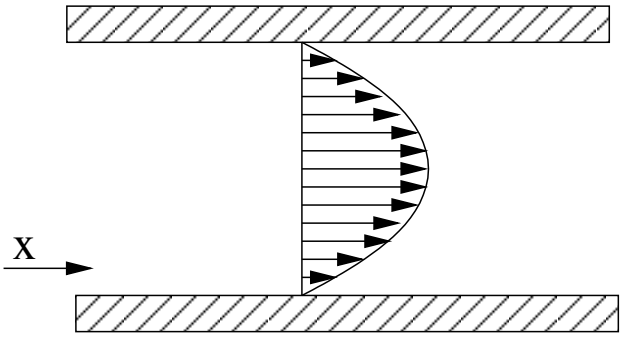


Figure 3.1: Poiseuille flow

The schematic domain can be seen in Fig. 3.2

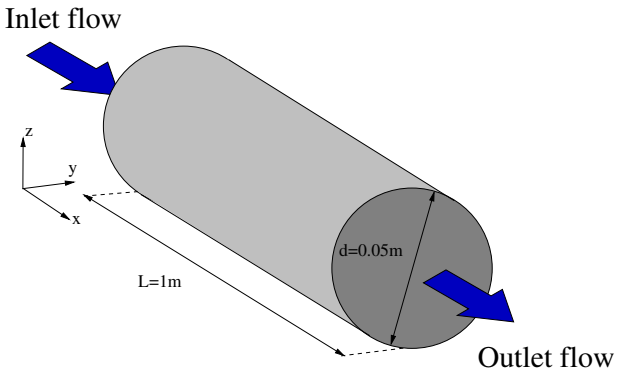


Figure 3.2: Circular pipe dimensions

The fluid used in this simulation is water, which has a very low kinematic viscosity ν. The pipe diameter is d. Hence, the simulation parameters are

$$\nu = 10^{-6} m^2/s \qquad d = 0.05m \qquad u = 0.002m/s$$

Thus, the Reynolds number is

$$Re = \frac{u\,d}{\nu} \longrightarrow Re = \frac{0.002m/s\,0.05m}{10^{-6}m^2/s} = 100$$

Different macroscopic boundary conditions are needed to simulate this flow

pattern. In the inlet and outlet of the pipe a homogeneous velocity profile is assumed, and the surrounding walls of the pipe are assumed to be non-moving without slip. This is translated to microscopic boundary conditions in the lattice framework as bouncing-back boundaries to simulate the pipe, and velocity boundaries to assume the homogeneous velocity at the inlet and outlet.

When a constant velocity is applied at the inlet, the flow needs some distance to develop its final shape, where the velocity field is not dependent on the longitudinal direction. This effect is described in Fig. 3.3.

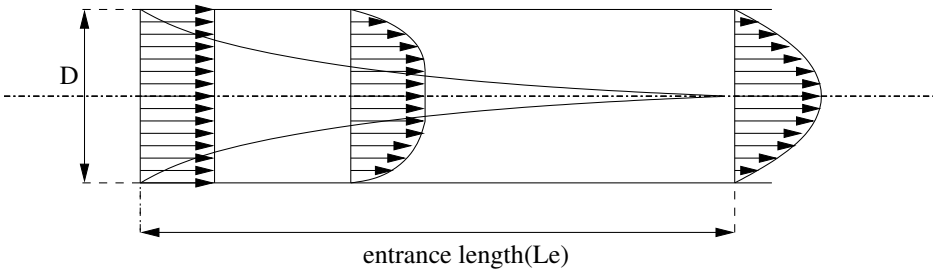


Figure 3.3: Entrance length effect

A flow with a Reynolds number below $\sim$ 2000 is laminar [CK04], thereby one can approximate the entrance length effect in a laminar flow regime by the following equation:

$$\frac{L_e}{D} \approx 0.06\, Re$$

Where L_e is the length of the pipe needed to develop the full profile and D its diameter. Therefore, a length of $L = 20\, D$ has been chosen to prevent that the entrance effect avoids a complete development of the velocity profile.
The characteristic discrete values in the lattice framework are

$$\nu_{lb} = 0.03 \qquad\qquad u_{lb} = 0.1$$

and the other parameters are calculated taking into account that the Reynolds

number must be maintained

$$Re = \frac{u_{lb}\, d_{lb}}{\nu} \longrightarrow d_{lb} = \frac{100 * 0.03}{0.1} = 30\, lu$$

Thus, since $l_{lb} = 20\, d_{lb} = 600\, lu$, the lattice has a length of $600\, lu$ and the diameter of the pipe is $30\, lu$.

After 3000 iterations to ensure that the system has reached a stationary solution, the results are shown in Figs. 3.4 and 3.5, where it can be seen, that the profile of the velocity field has a parabolic shape which matches with the correct shape of the Poiseuille flow as shown in Fig. 3.6.

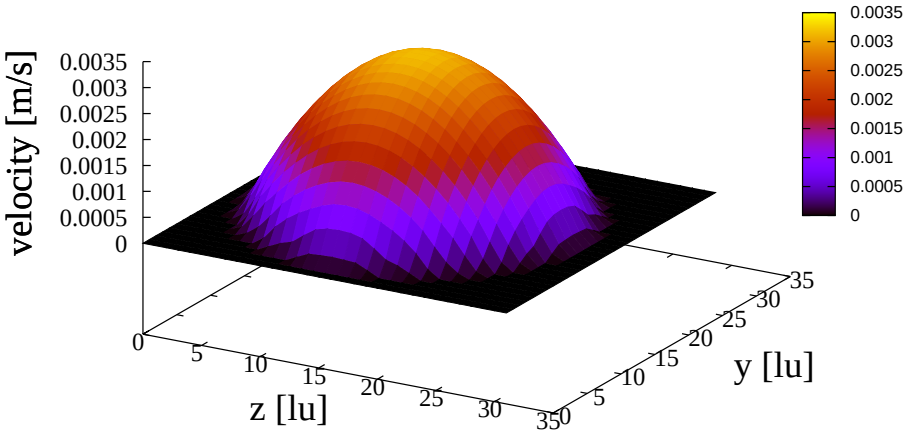


Figure 3.4: Velocity field at $x = 300\, lu$

Analysis of results

In references [SJ06][CK04], the velocity profile in a circular pipe follows this equation,

$$u(x) = U_{max}\left(1 - \frac{a^2}{R^2}\right) \tag{3.1}$$

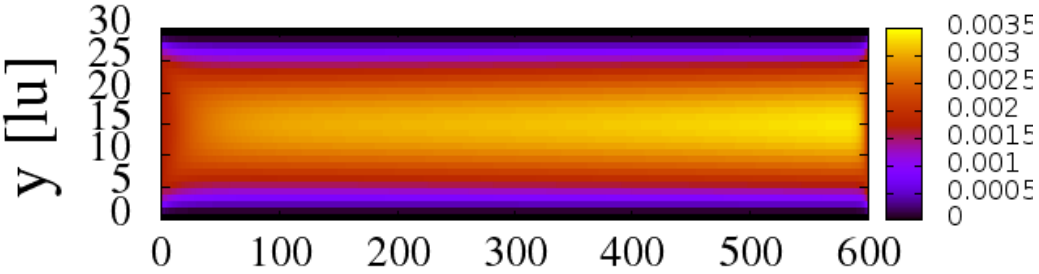


Figure 3.5: Velocity field at $z = 15\,lu$

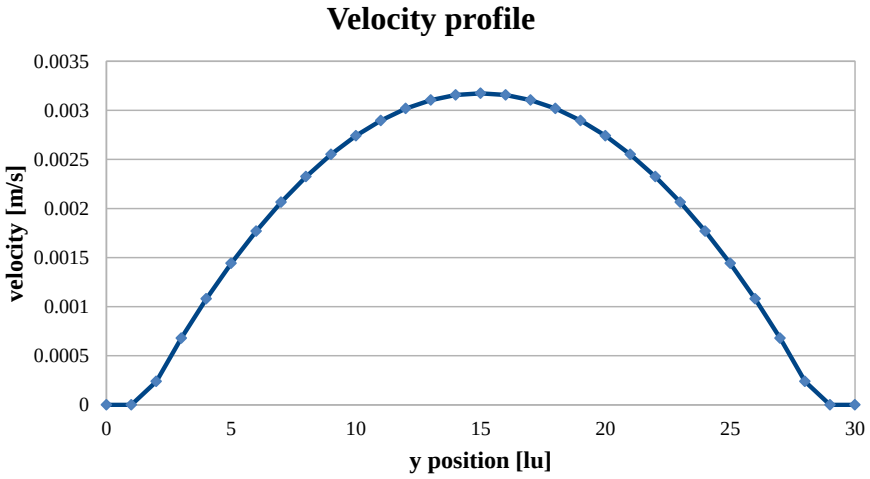


Figure 3.6: Flow profile perpendicular to the flow direction and along a straight line through the pipe center line

which can be derived from the NS equations. Hence, it is clear that the theoretical flow will have the form of a parabola proportional to some constant. This constant is the U_{max}. In Eq. (3.1) U_{max} can be related in the NS

framework to the pressure gradient by

$$U_{max} = \frac{dp}{dx}\frac{R^2}{4\mu}$$

Since the *Lattice Boltzmann Equation* is more fundamental than the NS equations, the velocity profile resulted from the LBM should agree with the parabolic profile from (3.1). It can be observed in Fig. 3.6
Thereby, the maximum velocity of the cut in $x = 300\,lu$, that can be seen in Fig. 3.6, will be used to approximate the parabolic shape and validate the LBM.

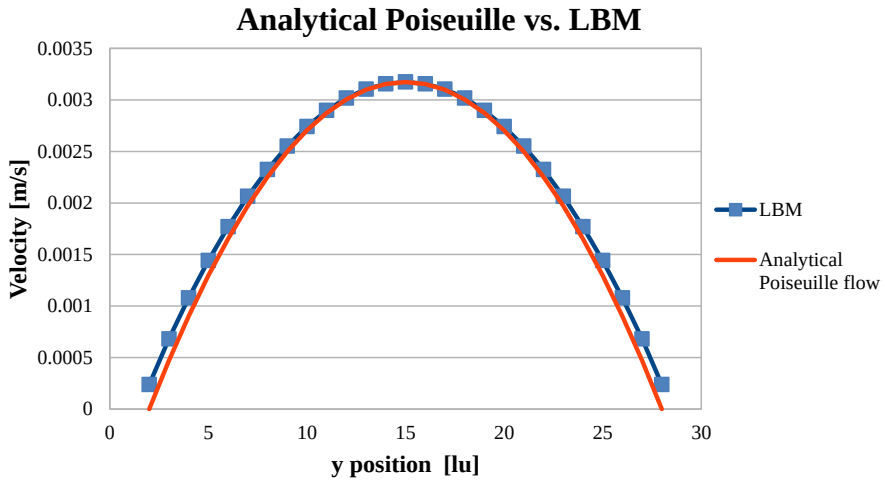


Figure 3.7: Comparison of the theoretical parabola shape with the experimental velocity profile

In Fig. 3.7 the comparison between the theoretical parabola shape and the velocity profile obtained applying the results can be seen. The relative error can be seen plotted in Fig. 3.7, and one can observe that the error increases dramatically when the flow approaches the boundary. This could be due several reasons.

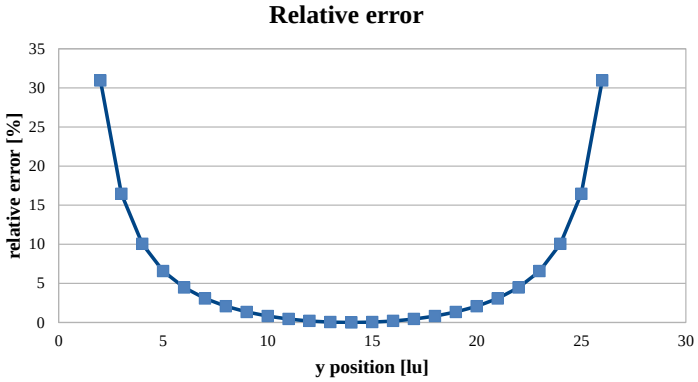


Figure 3.8: Relative error

1. The inaccuracy introduced by the bounce-back boundaries. It is known that bounce-back boundaries are an easy-to-implement tool which behaves correctly in the simulation. The price paid for this simplicity is a certain error introduced in the system. It can be seen that the boundaries introduce a small velocity that deviates the velocity profile from its theoretical value. This is due to the fact that the boundaries are placed between two nodes, then [Sat10] the bounce-back boundary does not respect the correct timing to reach the wall and come back to the original starting point. This is why the velocity is introduced, and therefore, this is the main cause of error.

2. The first reason is worsened when the lattice dimensions are not large enough, and thereby, the geometric accuracy is not the best. Defining a circumference in an area of $30\,lu \times 30\,lu$ gives a certain circumference shape, but far from a perfect circumference. It can be improved using a more accurate version of the boundary conditions called *YMLS* bounce-back boundaries [Sat10].

In addition to the velocity profile analysis, the entrance length effect can also be studied with the data generated. The entrance length is the distance that the flow takes to develop its profile from a constant velocity profile to a full developed profile (*i.e.* parabolic profile in the Poiseuille flow case). This

effect is schematically shown in Fig. 3.3

As said before, this entrance length (L_e) can be approximated for laminar flows by $\frac{L_e}{D} = 0.06\,Re$, which in this case is $\frac{L_e}{D} = 6$. Therefore, with the factor $\frac{L_e}{D} = 20$ chosen previously one should be able to observe the entrance length effect.

In Fig. 3.5 where the velocity field is plotted for $z = 15$, one can see a sort of evolution in the velocity field forming a cone, but it is not clear when the velocity profile is fully developed. Therefore, the graphic in Fig. 3.9 has been plotted with the data generated with different cuts.
It is easy to see that in the coordinates near the entrance the velocity profile

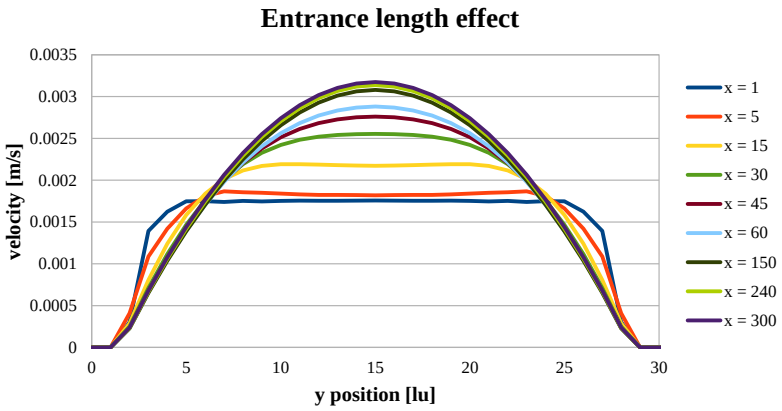


Figure 3.9: Development of the velocity profile in function of x

is square-shaped, as it should be due to the uniform velocity at the entrance, and as the flow advances through the pipe, it develops into the parabolic shape. One can observe that between $x = 150$ and $x = 240$ the flow develops poorly, so one can consider that the entrance length is between these two values. Thereby, one can conclude that the entrance length effect is correctly

modelled by the LBM.

3.2 3D Couette flow in a squared pipe.

The Couette flow is another flow pattern, which focuses on different class of flow problems. Therefore different macroscopic boundary conditions are needed. In this case a rectangular pipe is used to simplify the introduction of velocity in the wall.

3.2.1 Plane Couette flow with periodic boundaries.

The Couette flow is a specific flow driven by the motion of a plane over another one [CK04]. This can be seen schematically in Fig 3.10. The specific case of the plane Couette flow is the one where no pressure gradient is applied in the system.

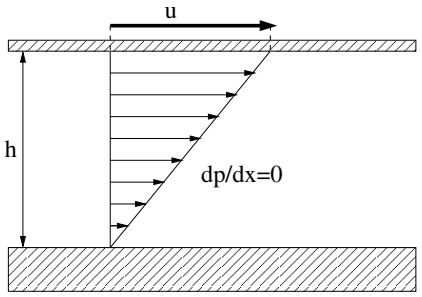


Figure 3.10: Plane Couette flow scheme

Periodic boundaries are implemented to study the effect of a lid velocity in the wall. Thereby, the domain is as shown in Fig. 3.11

The simulation parameters are chosen as

$$\nu = 10^{-6} m^2/s \qquad h = 0.05m \qquad u_{wall} = 0.002m/s$$

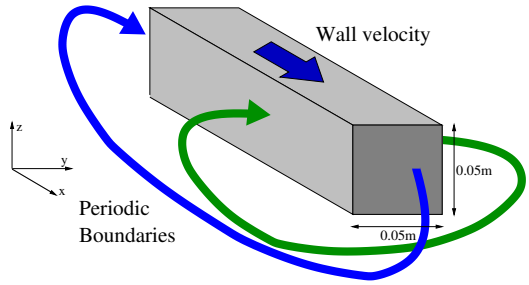


Figure 3.11: Scheme of the Couette domain

Hence, the Reynolds number is defined

$$Re = \frac{0.002m/s\ 0.05m}{10^{-6}m^2/s} = 100$$

Since these parameters are the same as the previous test, the lattice is defined by the same width of $30lu$, but due to the periodic boundaries the length is limited to $60lu$. Therefore, the general dimensions of the lattice are $60lu \times 30lu \times 30lu$. After sufficient time iterations to ensure a stationary stat (as expected from analytic calculations using the NS equation), we obtained the solution, the shape of which is shown in Fig. 3.12.

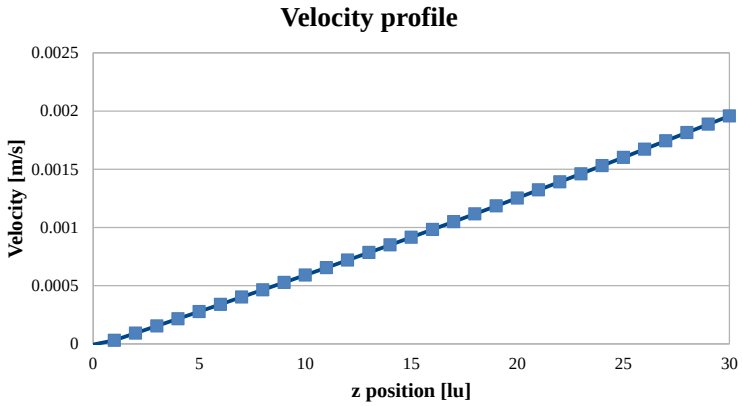


Figure 3.12: Velocity profile at x=30lu

Analysis of results.

It is easy to see that the result is a lineal evolution of the velocity field, which is expected from the theoretical Couette flow.
Also, if one observes the velocity field, it is clear that the lid condition on the upper wall is well simulated, as the velocity of the wall and the layer of fluid in contact with it is the same.

In this exact case the velocity only depends on the z coordinate. Therefore, theoretically $u(0) = 0$ and $u(z = 30lu) = u_{wall} = 0.1$. Thus, the theoretical velocity has this linear relation:

$$u(z) = u_{wall}\frac{z}{h}$$

In Fig. 3.13 the theoretical values are plotted in comparison with the simulation results. One can observe that the results follow the expected theoretical values.

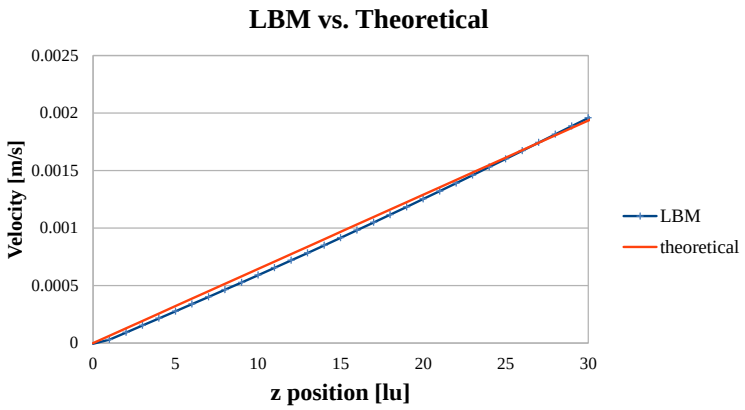


Figure 3.13: Theoretical plane couette flow velocity values in comparison with the results of the simulation.

3.2.2 Couette flow with pressure gradient.

To introduce a macroscopic pressure gradient in the system the microscopic boundary conditions must be modified. The periodic boundary conditions used in the previous subsection are replaced by velocity boundary conditions to introduce a pressure gradient. Thus, one can observe how the flow evolves when a velocity is introduced in addition to the lid velocity. This lid velocity will have in one case the same sense as the velocity introduced by the boundaries, and in the other case the lid velocity will have the opposite direction. This can be seen in Fig. 3.14. The domain of the system is the

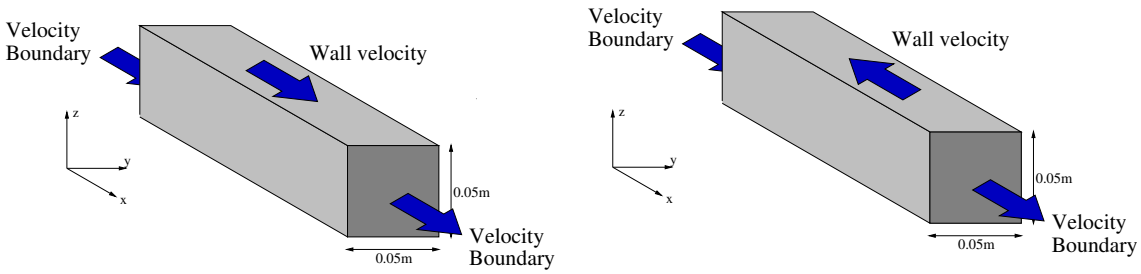


Figure 3.14: Different application of wall velocity condition

same as in subsection 3.2.1. Therefore, the simulation parameters and the lattice dimension are the same ($300\,lu \times 30\,lu \times 30\,lu$), where the length of the domain has been increased because there are no periodic boundaries.

If velocity in the wall is introduced to a system that has a pressure gradient (*i.e.* microscopic introduced velocity), the flow pattern depends on the sign of this pressure gradient [CK04]. This can be seen in Fig. 3.15.

Therefore two different cases are studied.

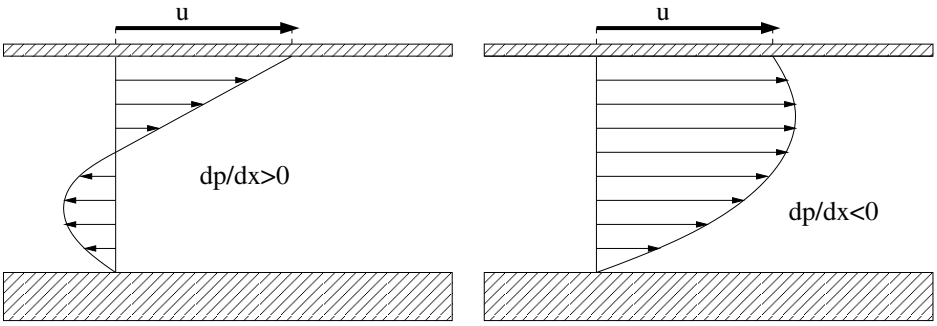


Figure 3.15: Couette flow with a pressure gradient

Couette flow with negative pressure gradient

The following relation between the modulus of the different velocity boundary conditions is inposed by the boundary conditions:

$$u_{wall} = 2\,u_{inlet} = 2\,u_{outlet} = 0.002\,m/s$$

Therefore the results can be seen in Fig. 3.16.

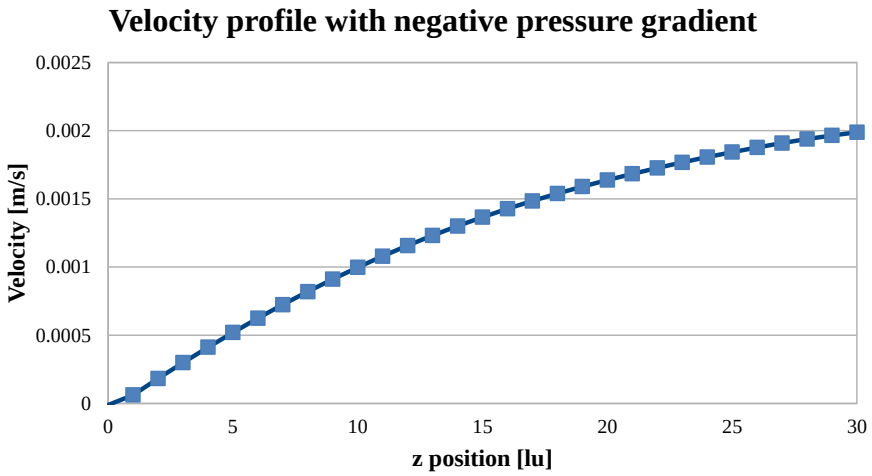


Figure 3.16: Velocity field with negative pressure gradient

Here, one can see that the flow profile has the same shape as the ideal case shown in Fig. 3.15.

To test the accuracy of the implementation one can decompose the Couette flow with a pressure gradient in a lineal combination of two other flow patterns (3.17), the Poiseuille flow and the Plane Couette flow. Thereby, if the

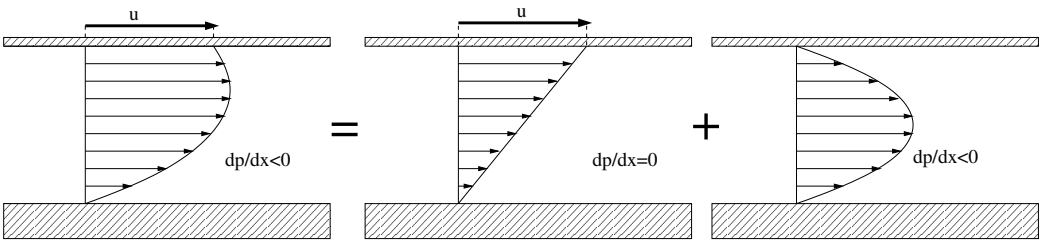


Figure 3.17: Lineal decomposition of the Couette flow

flow profile in Fig. 3.16 is decomposed the same way, the result is as expected (Fig. 3.18)

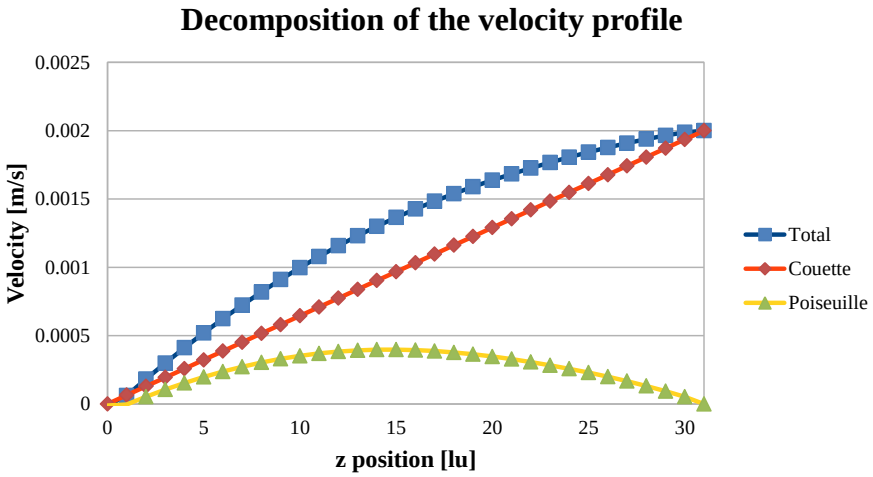


Figure 3.18: Decomposition of the results

It is easy to see that the flow pattern decomposed follows the theory. This test is composed from a superposition of the Poiseuille flow introduced by

the microscopic velocity boundaries and the Plane Couette flow introduced by the velocity of the wall. In addition, one must notice that the parabola has its maximum lower than one could have expected. This is because of the influence of the lid velocity which is rather large in comparison to the velocity introduced ($u_{wall} = 2u_{inlet}$).

Couette flow with positive pressure gradient

In this case, the following relation between the modulus of the different velocity boundary conditions is inposed by the boundary conditions:

$$u_{wall} = -2\,u_{inlet} = -2\,u_{outlet} = -0.002\,m/s$$

The same procedure as above is followed to analyse the results of the Couette flow with $\frac{dp}{dx} > 0$. The flow profile is as expected by the theory, and one can

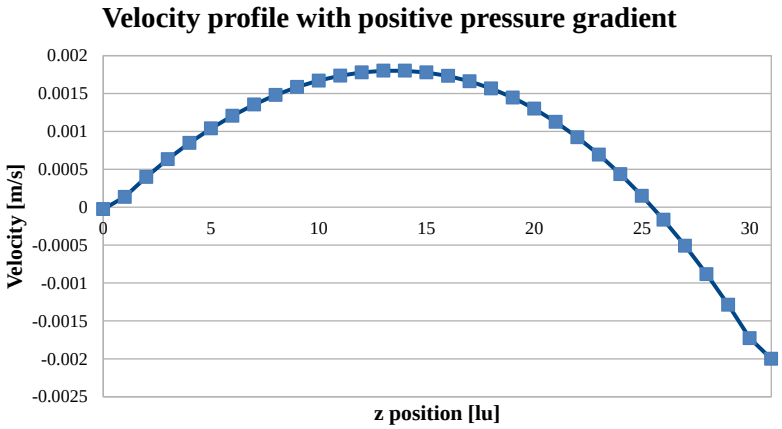


Figure 3.19: Velocity profile resulted from the simulation of the Couette flow with a positive pressure gradient

see that in Fig. 3.19. If the flow profile is decomposed, one can see in Fig 3.20 that the lid velocity in the wall produces the same effect as in the previous case. The parabolic profile is slightly deviated from the centre due to the presence of the large lid velocity in the wall.

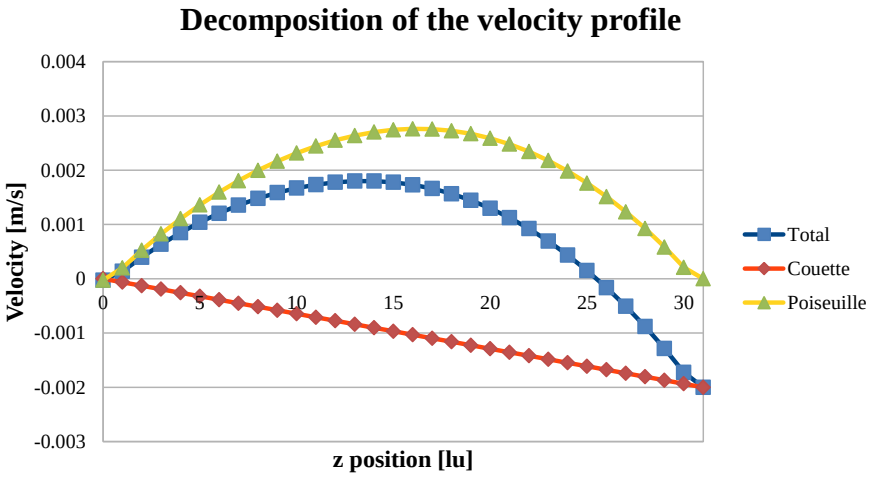


Figure 3.20: Decomposed velocity profile

3.3 Lid driven cavity

The lid driven cavity is a qualitative benchmark useful to test the behaviour of the fluid. The basic scheme of this test can be seen in Fig. 3.21 In this

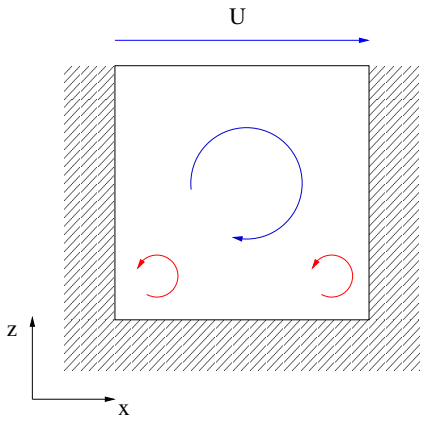


Figure 3.21: Scheme of the lid driven cavity test

benchmark different flow regimes will be tested by changing the Reynolds number of the system. The common characteristics values between this flow

regimes are:

$$\nu = 10^{-6} m^2/s \qquad\qquad l = 0.05m$$

The flow pattern (characterized by the Reynolds number) will vary when the characteristic velocity (wall velocity) changes. Therefore, these are the following parameters for the different simulations.

Re = 100

$$u_{wall} = 0.002m/s \qquad l = 0.05m$$
$$u_{lb} = 0.02lu/ts \qquad N_x = 50lu \qquad \nu_{lb} = 0.01$$

Re = 500

$$u_{wall} = 0.01m/s \qquad l = 0.05m$$
$$u_{lb} = 0.1lu/ts \qquad N_x = 50lu \qquad \nu_{lb} = 0.01$$

Re = 1000

$$u_{wall} = 0.02m/s \qquad l = 0.05m$$
$$u_{lb} = 0.2lu/ts \qquad N_x = 50lu \qquad \nu_{lb} = 0.01$$

Re = 3000

$$u_{wall} = 0.06m/s \qquad l = 0.05m$$
$$u_{lb} = 0.2lu/ts \qquad N_x = 75lu \qquad \nu_{lb} = 0.005$$

Analysis of results

The lid driven cavity test is expected to show some differences in the fluid behaviour when the Reynolds number increases [Mel13].

In the results listed in Figs. 3.22, 3.23, 3.24 and 3.25 one can see that the expected behaviour is simulated.

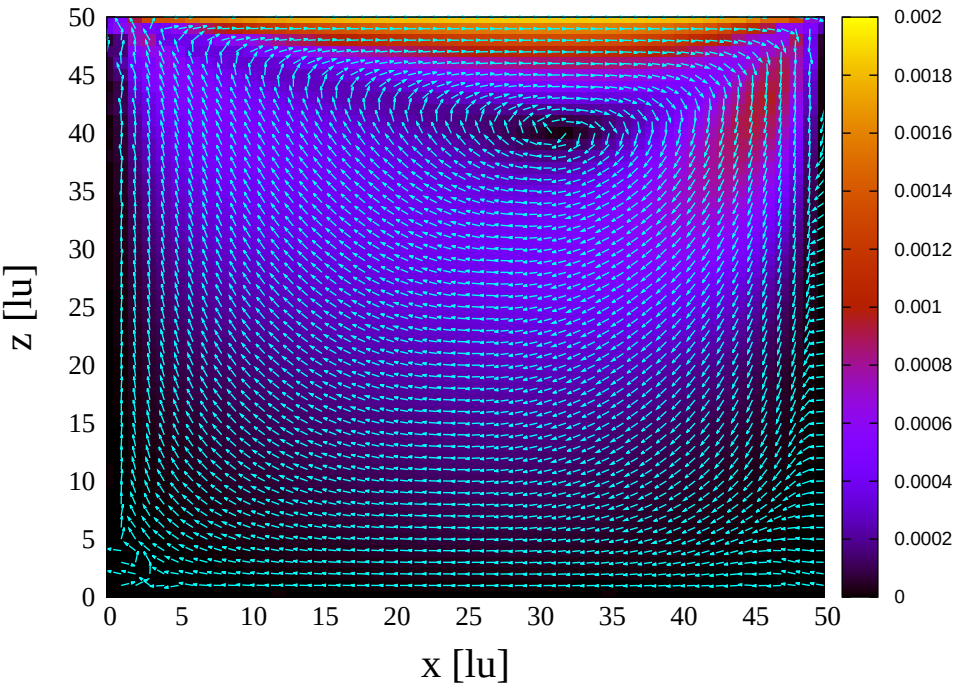


Figure 3.22: Lid driven cavity, $Re = 100$

For $Re = 100$ flow pattern, the lid velocity creates a vortex in the upper part of the cavity. This vortex has an elongated shape and later it will evolve to a more centred position when the Reynolds number increases.
In addition one can see that the velocity modulus increases at the upper-right side of the cavity.

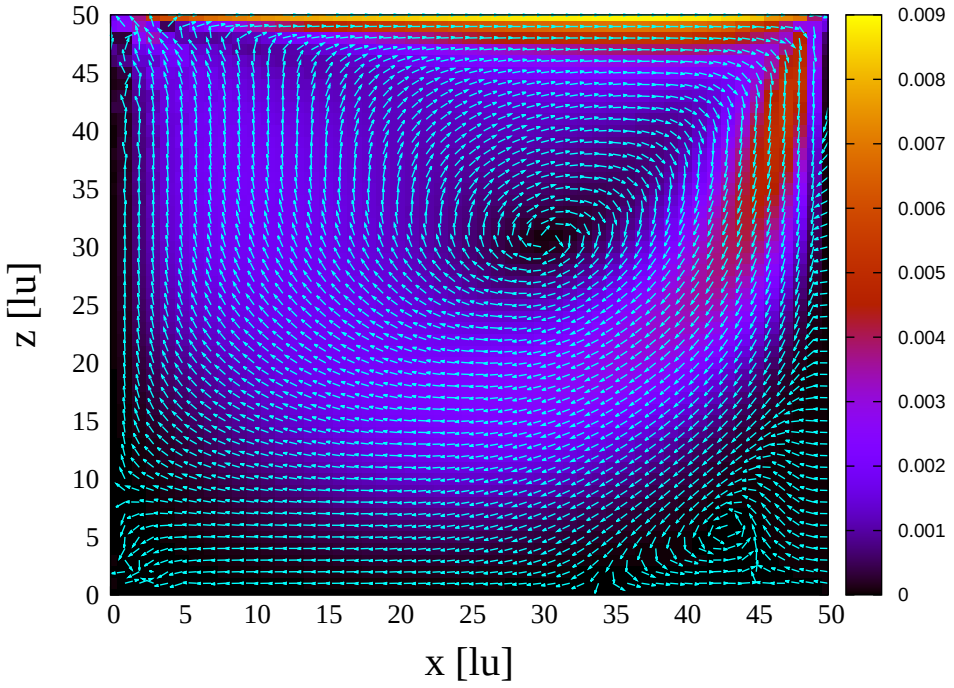


Figure 3.23: Lid driven cavity, $Re = 500$

For $Re = 500$ one can see that the previous vortex has a centred position and it has a more rounded shape. Moreover a vortex in the lower-right corner has been developed, and instabilities in the lower-left corner are evolving.

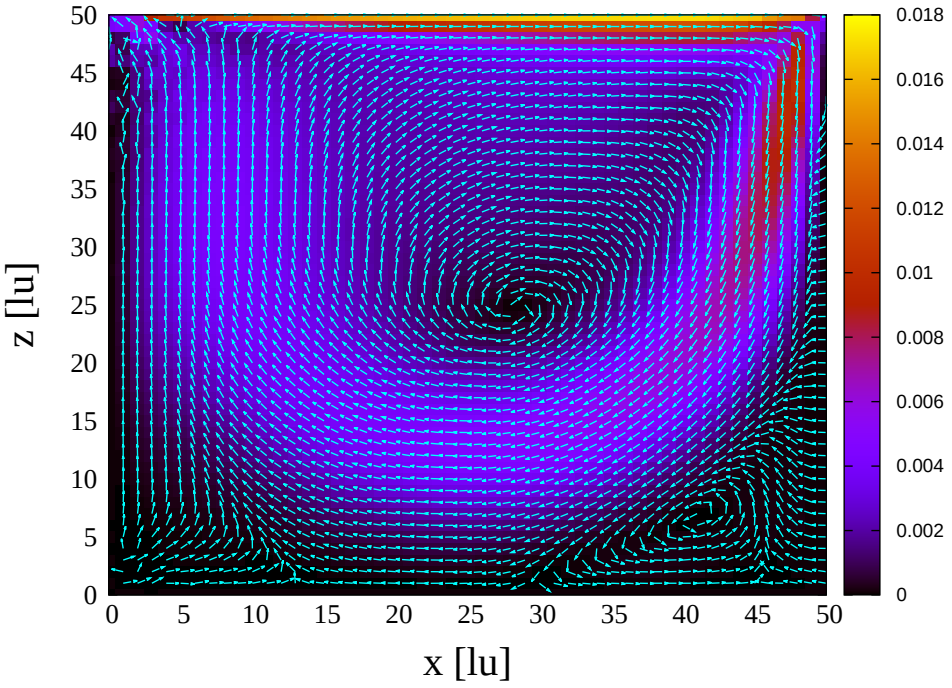


Figure 3.24: Lid driven cavity, $Re = 1000$

When $Re = 1000$ the central vortex is even more centred, and the vortex in the low-right corner is still present. A vortex in the low-left corner has begun to develop.

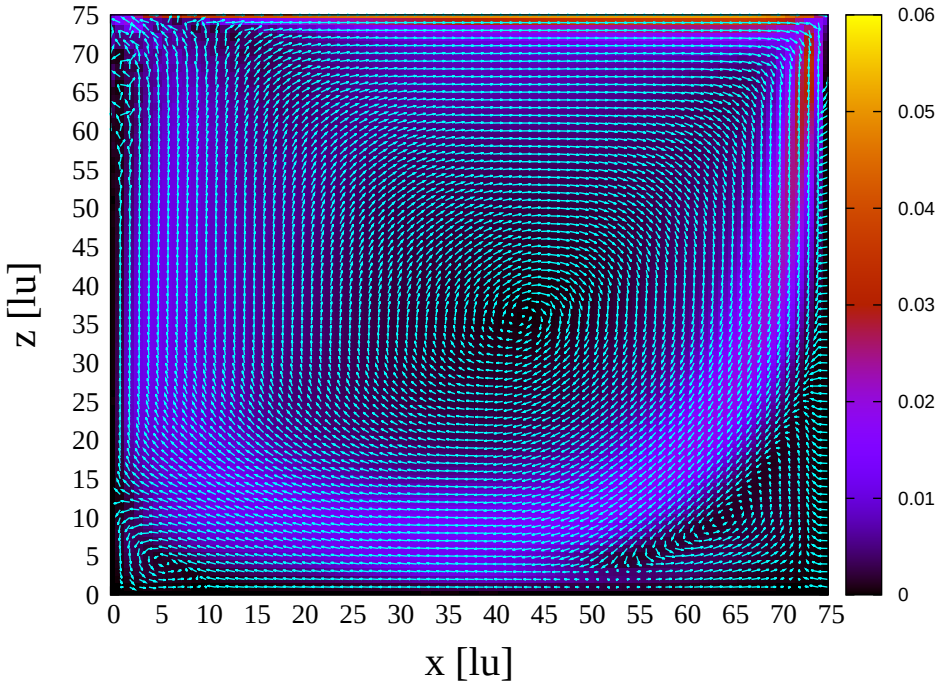


Figure 3.25: Lid driven cavity, $Re = 3000$

For $Re = 3000$ it can be seen that the central vortex and the two lower eddies are there still, but the central vortex has increased its size. In addition, a vortex in the upper-left corner is forming.

3.4 Flow around an obstacle

Another qualitative benchmark is to observe how the flow evolves when an obstacle is set inside the domain.
For the cases, when the Reynolds number is small, the flow pattern must show a laminar flow pattern. When the Reynolds number increases and exceeds a critical Reynolds number, this flow pattern should not be laminar anymore, and become unstable, with the formation of vortices in the vicinity of the obstacle.

To test this behaviour a cilindrical obstacle is submitted to a homogeneous flow, inside a rectangular domain, varying the Reynolds numbers of the system. The scheme can be seen in Fig. 3.26.

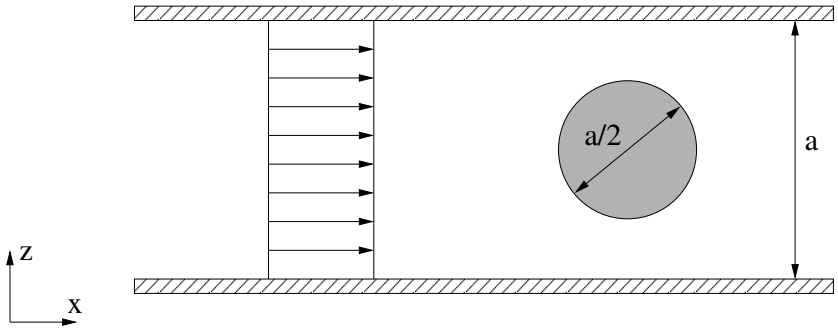


Figure 3.26: Domain scheme of the flow around an obstacle test

There have been done 3 simulations, using water as the fluid ($\nu = 10^{-6} m^2/s$), varying the Reynolds number, and thus, the flow pattern. The characteristic values of every simulation are:

Re = 100

$$u_{avg} = 0.002 m/s \qquad a = 0.05 m$$
$$u_{lb} = 0.02 lu/ts \qquad N_z = lu \qquad \nu_{lb} = 0.01 \qquad 5000\ iterations$$

Re = 500

$$u_{avg} = 0.01m/s \qquad a = 0.05m$$
$$u_{lb} = 0.1lu/ts \qquad N_z = 50lu \qquad \nu_{lb} = 0.01 \qquad 5000\ iterations$$

Re = 1000

$$u_{avg} = 0.02m/s \qquad a = 0.05m$$
$$u_{lb} = 0.1lu/ts \qquad N_z = 50lu \qquad \nu_{lb} = 0.005 \qquad 5000\ iterations$$

3.4.1 Analysis of results

The results are shown in Figs. 3.27, 3.28 and 3.29. In the case of $Re = 1000$ the same discrete characteristic velocity as the case of $Re = 500$ has been used, but the viscosity in the lattice framework was set to the half to modify the flow regime.

Low Reynolds number

When the flow is characterized by a low Reynolds number, the velocity field around the cylinder increases its magnitude due to the Bernoulli effect, but the flow is maintained laminar and no vortices are created. In addition, it is observable that the velocity decreases at the front point of the cylinder which is the point of maximum pressure, and in the rear point the velocity also decreases, where the pressure is minimum. This can be observed in Fig. 3.27

Intermediate Reynolds number

In this case the incoming flow regime is still laminar, but the velocity is higher, thereby, the fluid interacts differently with the obstacle. It is clearly observable in Fig. 3.28 that in the obstacle front point the fluid has a decrease of velocity, so it has the same behaviour as in the previous case, but behind the cylinder two elongated eddies have

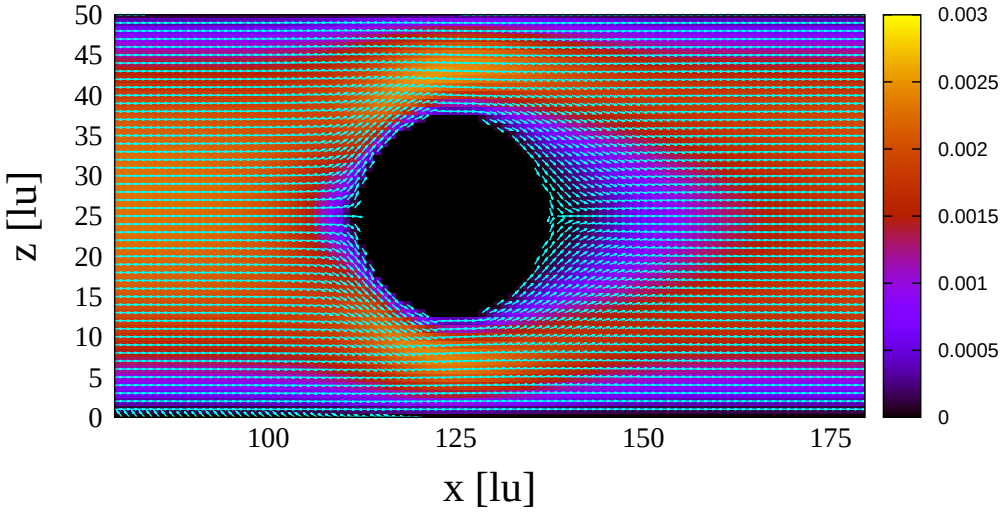


Figure 3.27: Low Reynolds flow interaction with an obstacle

appear. This eddies are more elongated the more the Reynolds number increases, and they change the velocity direction inside them, but over their limits the flow pattern is still laminar.

Higher Reynolds number

When Reynolds number has increased significantly, the fluid in front of the obstacle behaves as the other cases because the flow is still laminar. Moreover, the part beyond the cylinder becomes unstable and creates what it is called *Von Karman Vortex Street* [CK04]. This phenomenon is a sequence of periodic eddies created due to the periodic fluctuation of the velocity direction and magnitude beyond the cylinder. In Fig. 3.29 one can see that a lower eddy has been developed beyond the cylinder (more or less at $x = 75lu$), and a second eddy in the upper part, just beyond the cylinder, is still developing.

After the verification that the different simulations match their theoretical behaviour, it can be said that LBM models correctly the interaction of the fluid with obstacles.

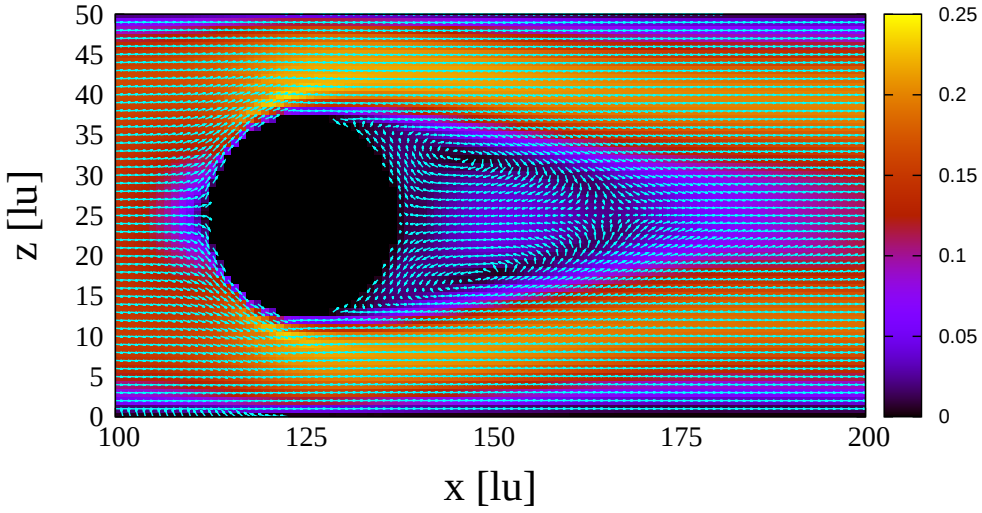


Figure 3.28: Intermediate Reynolds flow interaction with an obstacle

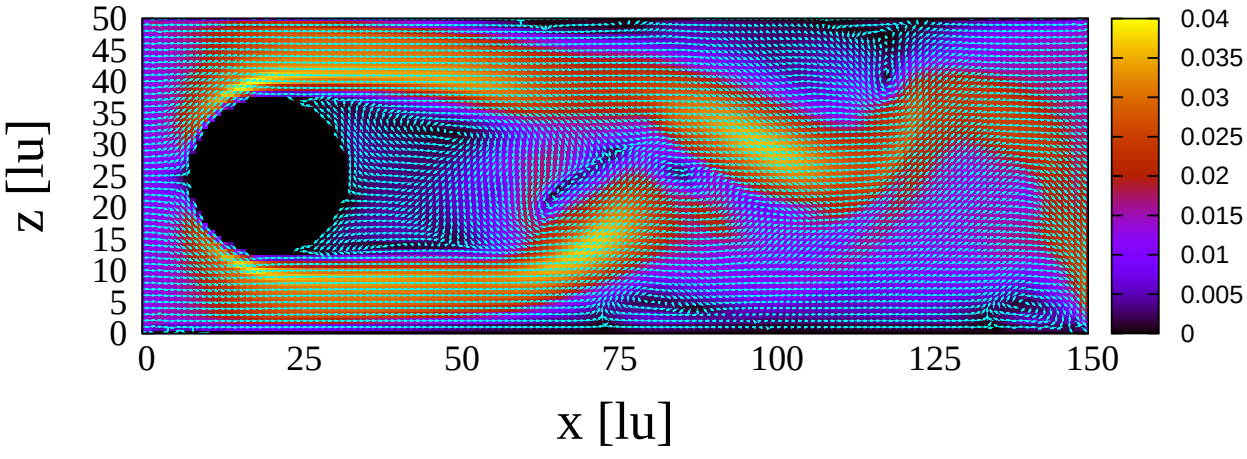


Figure 3.29: Higher Reynolds flow interaction with an obstacle, where the formation of *Von Karman Vortex Street* can be observed

Chapter 4

Application of the LBM

4.1 Lower plenum of a PWR

In the design of nuclear reactor cores, to know and predict the fluid behaviour is a critical issue, and different methods are used to simulate its behaviour under certain conditions. In this thesis, the lower plenum of a Pressurized Water Reactor (PWR) core is the subject of study. The scheme of the PWR vessel and its parts can be seen in Fig. 4.1.

Schematically, it is shown that the coolant enters through the inlet nozzle, and follows the downcomer, which is the space between the reactor vessel and the core barrel, to reach the lower plenum (see Fig. 4.2). There, the coolant flows upstream through the reactor core, where it exchanges the heat used to produce electricity afterwards.
The scope of this study is the coolant flow from the entrance channel until it flows through the core, neglecting temperature fluctuations (and possible phase transitions). It has been chosen this specific part of the nuclear reactor core because it would be interesting to study the flow pattern in the lower plenum. Furthermore, the coolant disturbances in the lower plenum can affect to the neutronic behaviour, which is also interesting.

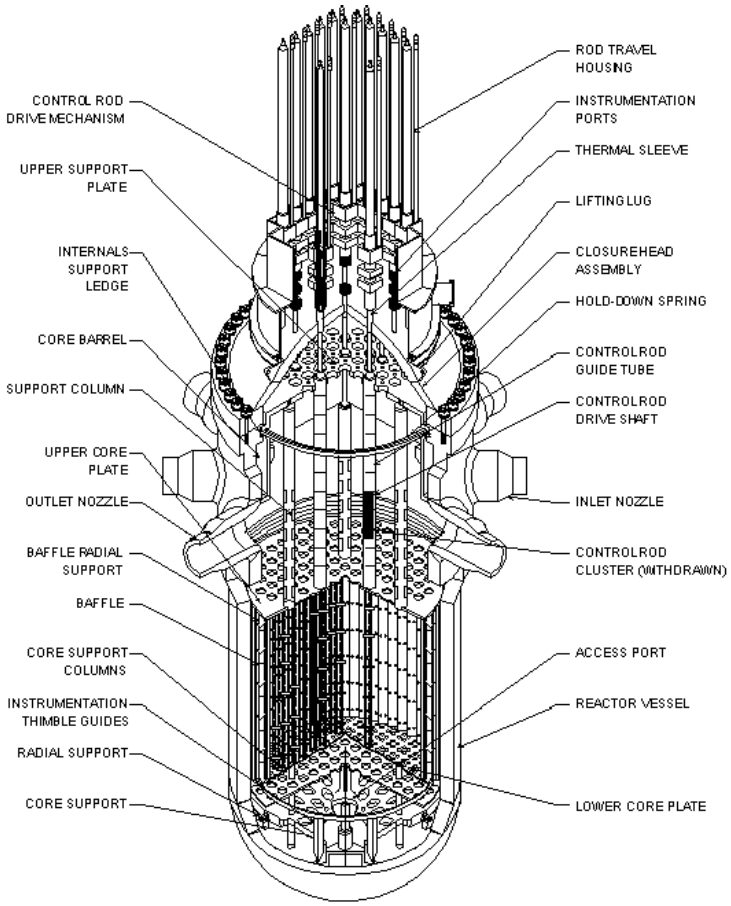


Figure 4.1: PWR core vessel scheme [Wikb]

To be able to simulate the lower plenum, a specific geometry must be defined to represent this part of the core. The dimensions of the geometry used can be seen in Fig. 4.3 .
One can observe in comparison with Fig. 4.1 that this is a simplification of a real lower plenum and it is far from the exact core geometry for the sake of investigating LBM for such an application.
The first step to apply the LBM is to define the domain and discretize the physical values. In PWR reactor types, the coolant used is water which has rather low kinematic viscosity $\nu_{h_2o} = 10^{-6} m^2/s$. In addition, high velocities are reached in the lower plenum region. An average velocity of $u_{avg} = 0.5 m/s$ is used to calculate the flow pattern.

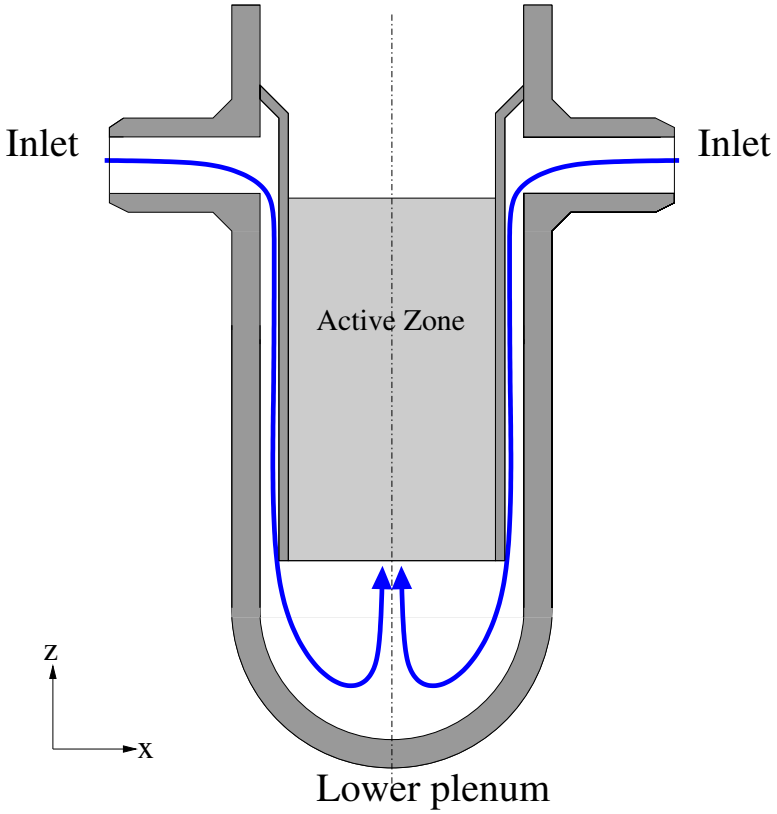


Figure 4.2: Lower plenum scheme

To simulate flows of larger Reynolds number, the order of thousands, a large amount of computer capacity is required, thus, a special computer is needed to perform the calculations, or post processing methods are used. An estimation of the computer requirements are listed in the appendix section A.1.4.

Due to the large requirements, a nested-scaling procedure has been used to simulate the lower plenum cavity. A first simulation has been made to obtain a average overall flow pattern. Thereafter, when the critical zones are located, a smaller cubic volume that contains these critical zones is considered. To increase its resolution, the velocity field values from the first resolution that were on the surface of this smaller cubic volume are mapped on its sides, and interpolated to increase its resolution.
The schematic procedure can be seen in Fig. 4.4. Therefore, the discrete

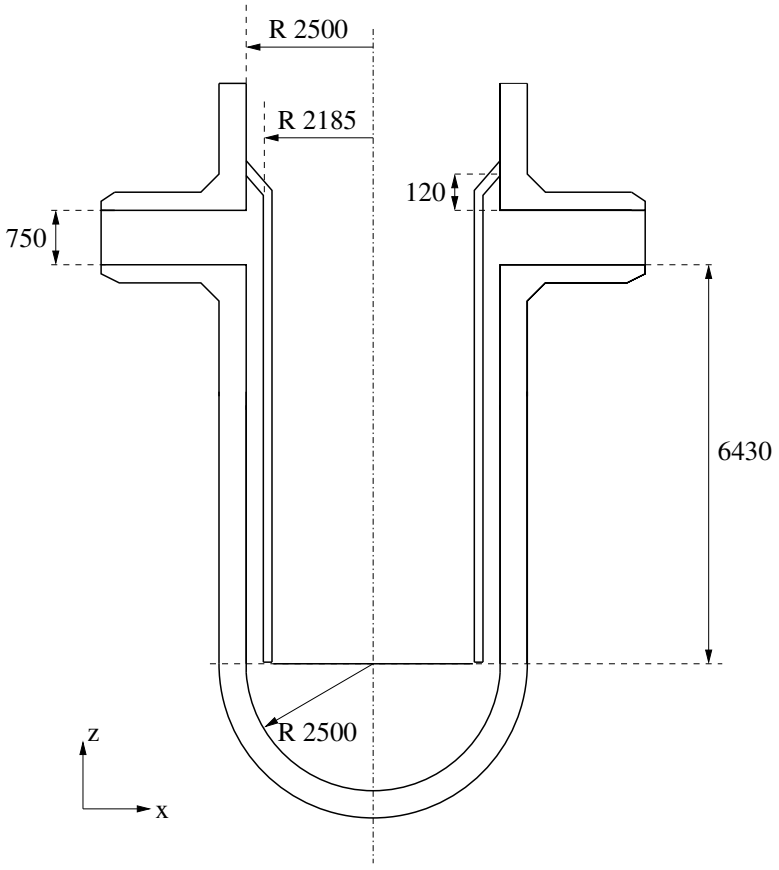


Figure 4.3: Lower plenum geometric scheme, units in mm

characteristic values used are

$$u_{lb} = 0.1 \qquad N_z = 133\,lu \qquad \nu_{lb} = \frac{0.02}{3}$$

The lattice dimensions are $106\,lu \times 73\,lu \times 133\,lu$. Velocity boundary conditions are applied at the entrance channel, and in the central region, where the flow passes through the core, a constant pressure boundary condition is applied. This pressure boundary induces a velocity in the boundary that must ensure together with the velocity boundaries that the conservation of mass condition is fulfilled.

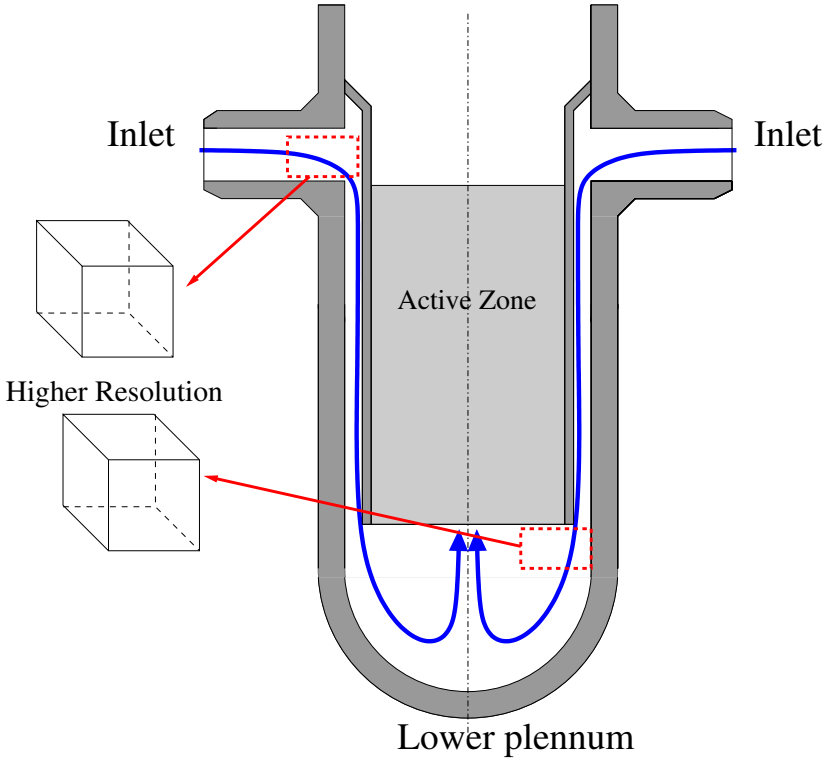


Figure 4.4: Multi-scaling procedure scheme

Therefore,

$$\dot{m}_{inlet,1} + \dot{m}_{inlet,2} = \dot{m}_{outlet}$$
$$(u_{inlet,1} + u_{inlet,2})A_{inlet} = u_{outlet}A_{oulet}$$

Thus, since the two inlet velocities at the entrance channel are imposed to be the same, u_{inlet} and u_{outlet} are related by

$$u_{inlet} = u_{outlet}\frac{A_{oulet}}{2A_{inlet}}$$

The results of this first simulation can be seen below

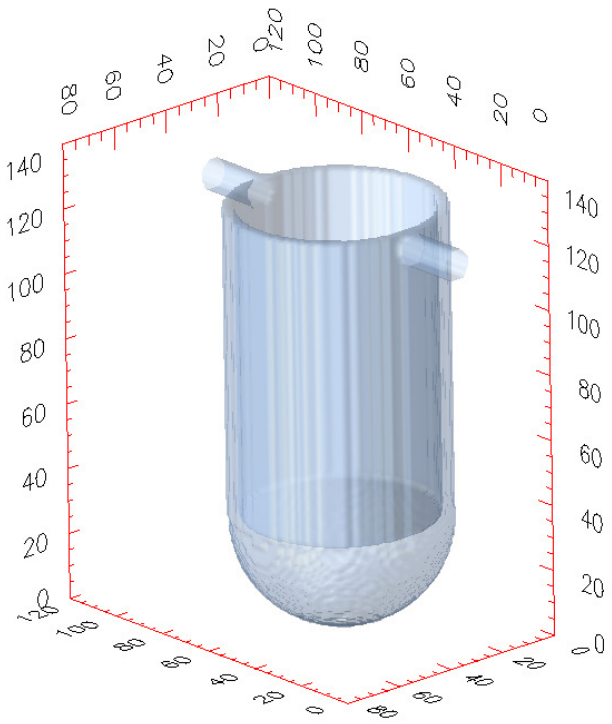


Figure 4.5: 3D geometry implemented

4.2 Analysis of results

The results of the first simulation of the lower plenum simulation are shown in Figs. 4.6, 4.7 and 4.8 One can see that the flow direction is the one expected, and the coolant enters from the inlet nozzle and flows through the downcomer to reach the lower plenum and it changes its direction and flows upstream to reach the core. In the junction between the entrance channel and the downcomer flow instabilities occur due to the abrupt change of direction.

In the downcomer region the flow velocity decreases rapidly due to the increase of volume available for the liquid. If the geometry had had more inlets nozzles, the velocity would not suffer this high decrease.

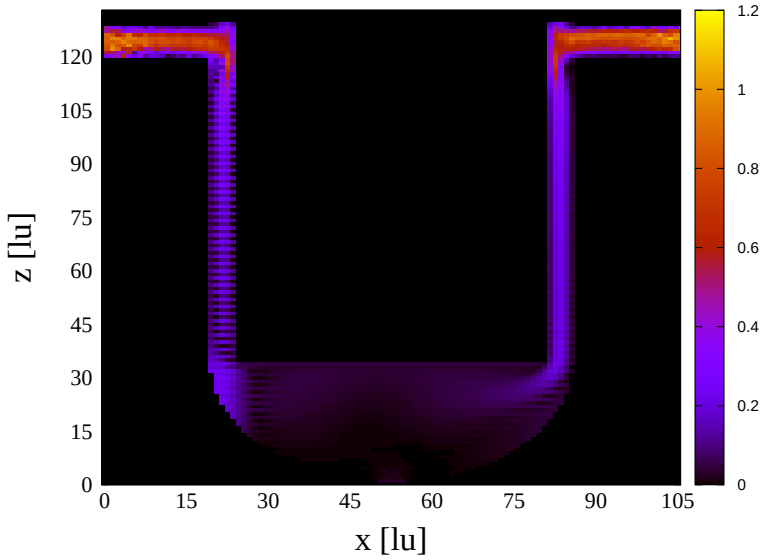


Figure 4.6: Velocity modulus (m/s) at $y = 37\,lu$

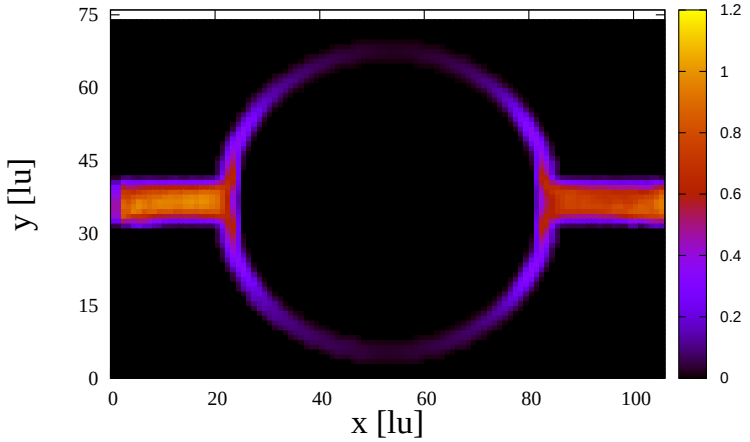


Figure 4.7: Velocity modulus (m/s) at $z = 123\,lu$

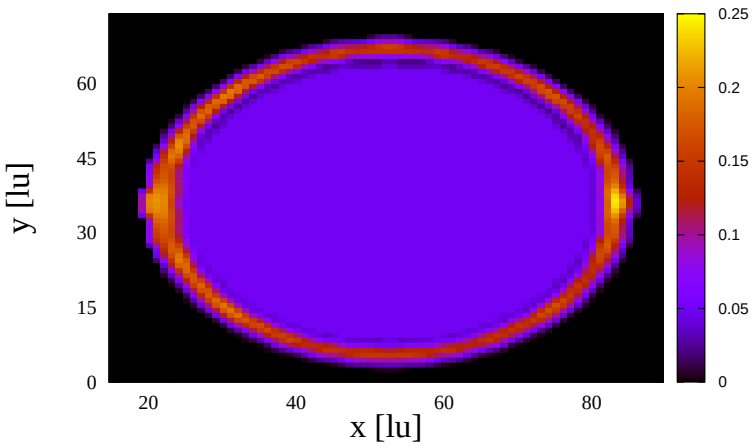


Figure 4.8: Velocity modulus (m/s) in the downcomer $(z = 34\,lu)$

First of all the mass conservation condition must be ensured in the lattice framework. To verify this, the velocities from the this framework at the inlet and outlet are plotted. One can see that the average velocity in the inlet

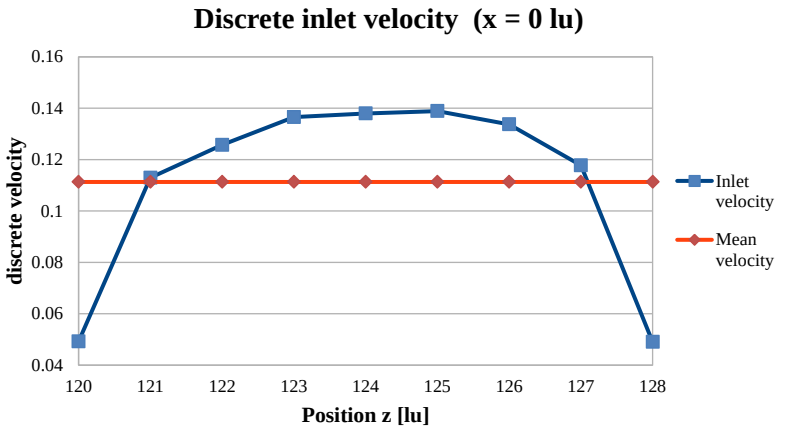


Figure 4.9: Velocity profile at the entrance channel, $x = 0\,lu$, $y = 37\,lu$

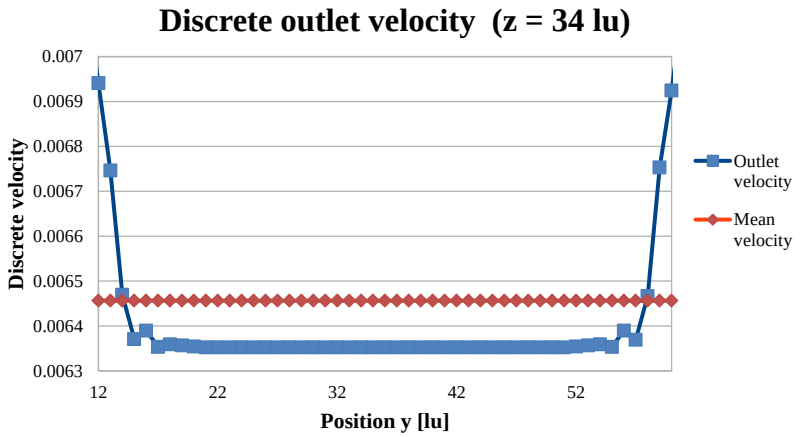


Figure 4.10: Velocity modulus at the outlet, $x = 56\,lu$, $z = 34\,lu$

and in the outlet are

$$u_{inlet} = 0.1113 \qquad\qquad u_{outlet} = 0.00645$$

Therefore, taking into account that there are two inlets, where the flow has the same velocity,

$$\dot{m}_{inlet,1} + \dot{m}_{inlet,2} = \dot{m}_{outlet}$$
$$2u_{inlet}A_{inlet} = u_{outlet}A_{outlet}$$

where, geometrically, from the dimensions in Fig. 4.3 it can be proved that

$$\frac{2A_{inlet}}{A_{outlet}} = 17$$

thus, this relation must be fulfilled

$$\frac{u_{inlet}}{u_{outlet}} = 17 \Longrightarrow \frac{0.1113}{0.00645} = 17.25$$

One can observe that the mass conservation can be considered fulfilled al-

though the velocity relation is slightly deviated. This deviation is caused by the poor resolution of the lattice, and with higher resolutions the velocity relations will be perfectly fulfilled.

If zoom is made in the two regions highlighted in Fig. 4.4, one can observe that instabilities occur in Figs. 4.11 and 4.12.

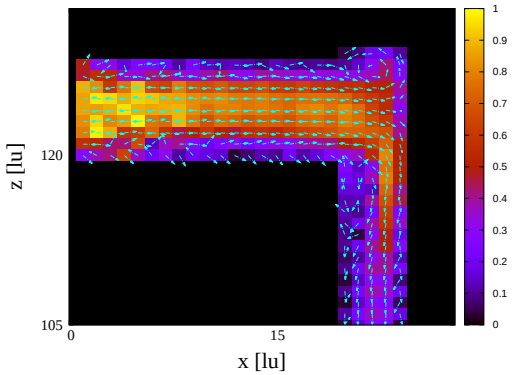


Figure 4.11: Velocity field in the junction between the entrance channel and the downcomer

The nested-scaling procedure is applied in those locations.
To be able to apply this procedure the no time-dependency of the solution must be ensured, which is after 5000 iterations. Therefore, a small cubic volume in the zone is considered, and the velocity in each plane of this cube is mapped. For the high resolution in this considered box, one needs also high resolution boundary conditions, which are taken from the coarser solution and are interpolated. A bilinear interpolation [bil] has been chosen to interpolate the values. The volume chosen has dimensions of $6\,lu \times 6\,lu \times 6\,lu$ and then, after the interpolation this volume has $24\,lu \times 24\,lu \times 24\,lu$.

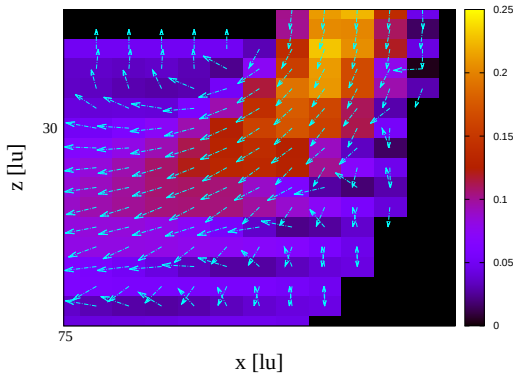


Figure 4.12: Velocity field in the junction between the lower plenum and the downcomer

The entrance channel and the downcomer junction

The result after this simulation with higher resolution can be seen in Fig. 4.13.

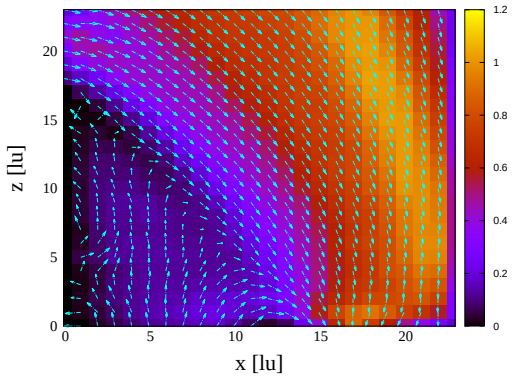


Figure 4.13: Result in high-resolution of the junction between the entrance channel and the downcomer.

In this case, when the resolution is increased, the plot shows that disturbances are created. One can see a large eddy in the outer wall side of the downcomer created due to the large difference of pressure. If the local density of every node in plotted, since density and pressure are related in the

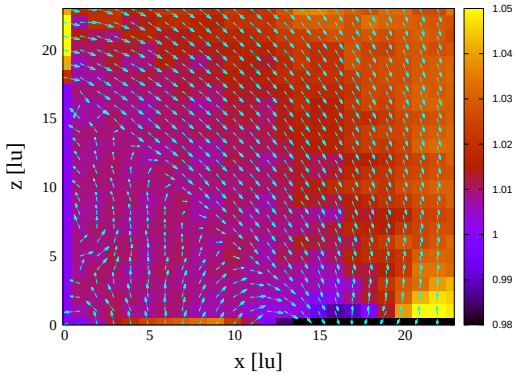


Figure 4.14: Density field in the entrance channel and downcomer junction.

lattice framework by its EOS, it can be seen that in the outer wall of the downcomer the pressure is lower than in the inner wall. This pressure gradient is what origins this velocity difference, and thus, these eddies are created.

Lower plenum.

The other zone where the multi-scaling procedure has been applied is the lower plenum. In this case the results can be seen in Fig. 4.15 where one

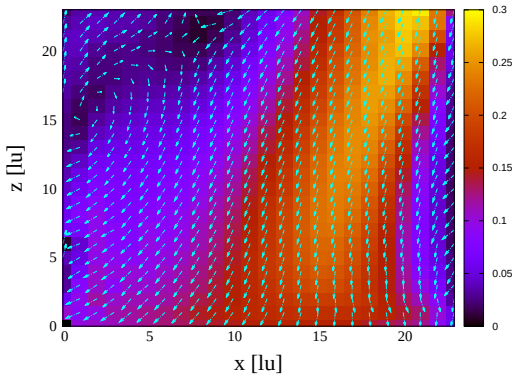


Figure 4.15: Multi-scaling result of the junction between the lower plenum and the downcomer.

can observe differences from Fig. 4.12. The first one is that velocity flows downstream more attached to the containing wall. In addition, one can see that eddies are formed in the top-left corner of the plot. This region corresponds to the region where the coolant flows upstream and leaves the lower plenum cavity.

Chapter 5

Conclusions

In this thesis the LBM was described and several applications of this method were implemented.

In chapter 2 the theoretical background of the method was introduced. First the Boltzmann distribution was derived, which is the distribution that holds the statistical treatment of the method. The Boltzmann equation was explained next. The main difficulty of this equation, the integral of the collision operator, is solved by the *BGK-Approximation* where this integral is substituted by a lineal term.
The discretization of the Boltzmann equation with the *BGK-Approximation* was introduced, all along with the lattice and the different lattice configurations, to define the lattice framework where the method is applied. Also in this framework the algorithm of the method was defined. The boundary conditions, are summarized also in this chapter, where one can see that they represent macroscopic conditions acting locally in every node.

In the second part of this thesis, applications of the model were implemented and tested, and their results were analysed.
In the first case, the Poiseuille flow was simulated in a 3D circular pipe. This simulation showed that the model is able to simulate the flow of a fluid cor-

rectly. In addition, one could see that the bounce-back boundary conditions introduce error in the system.

The Couette flow was also simulated. The results of this simulation showed that the LBM implemented simulate correctly the flow pattern, and it agrees with the theoretical results. When the Couette flow is applied together with the Poiseuille flow (Couette flow with pressure gradient), the method is able to simulate correctly this situation, as it was shown in the analysis of results.

The lid driven and the flow around an obstacle test showed that the model also simulates the interaction between the fluid and the obstacles correctly. In either test the fluid behaves differently depending on the flow regime, which is characterized by the Reynolds number. At higher Re numbers the flow is not laminar anymore and eddies and vortexes are created.

After those previous simulations, where it has been shown that the method simulates the fluid as it is expected, LBM was used to simulate a rather complex case. The simulation of the lower plenum of a PWR reactor core has been useful to remark several things. The first one, is that the simulation of flow regimes characterized by enormous Reynolds numbers is not entirely possible, at least without the access to special computers, with larger resources, or without post-processing treatments. Due to the limitation of both discrete velocity u_{lb} and discrete viscosity ν_{lb} in the lattice framework, the Reynolds number can only be enlarged by the lattice dimensions. To face this problem, a multi-scaling post-processing treatment has been used in this thesis. This procedure has been able to recreate the flow pattern in a small volume increasing the resolution of the first simulation results. Nonetheless it increases substantially the complexity of the simulation. Another remarkable point is that complex geometry can be easily implemented in the LBM, so it can be useful to simulate these situations that have complex geometry and not so high Reynolds numbers.

A general remark after the development of this thesis is that one can see that the boundary conditions are one of the most complex parts in the LBM. This is because the physical boundary conditions are, normally, macroscopic boundary conditions, and to transform these macroscopic conditions to the microscopic scale where the LBM works is not intuitive. In addition, the boundaries used in this thesis for the generation of geometry, the bounce-back boundaries, are one of the simplest schemes existing nowadays. Other more complex schemes for the boundary conditions can be used to increase the accuracy of the system. It must be said, also, that the complexity of the implementation increases dramatically, so one must take this into account.
In addition, this method works in the lattice framework, so the system can be re-scaled to a different system from the initial one while Re number is maintained. This gives LBM the enormous flexibility of solving one dimensionless problem and being able to re-scale it to several different physical problems.

Finally, an interesting observation considering the future perspectives of this method, is that LBM is intrinsically parallelizable. This is because the algorithm is applied locally on every node. Therefore, LBM will have a more important role in the future where multi processors platforms (either CPUs or GPUs) will take profit of this intrinsic parallelizable nature.

Appendix A

Code development

A.1 LBM implementation in C++.

In this appendix the implementation in `C++` of the model will be discussed. A general knowledge of `C++` and programing is supposed [C].

The implementation has been done using a class hierarchy taking advantage of the main property of the `C++` programing language. The design was thought to perform the method using vectors as cointainers of pointers lists to objects. It requires an abstraction effort during the design phase but it has an easier implementation in the programming phase.

A.1.1 Class hierarchy.

The classes implemented have a hierarchy and a relation between them. This relation is shown in Fig. A.1

Therefore, the classes used are listed and explained below.

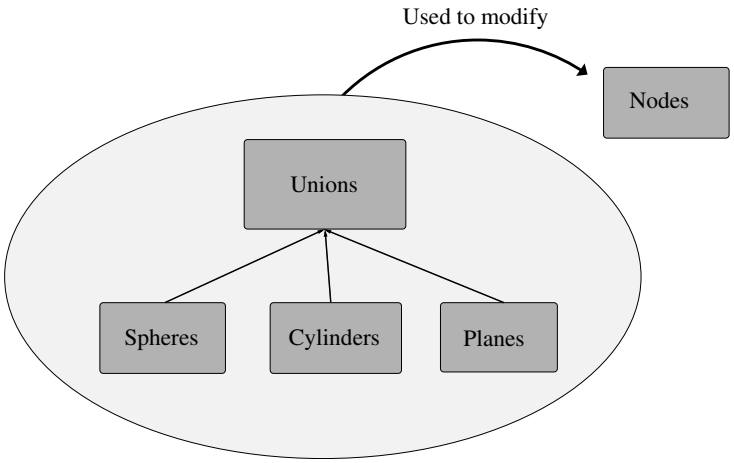


Figure A.1: Structure of the class implementation

Node Class

The node class is the basic class of the whole implementation and it has the following members and methods.

```
class Node {
  int i,j,k;                          //discrete node coord.
  float x,y,z;                        //physical node coord.
  float u,u_x,u_y,u_z;                //discrete velocities
  float up,u_xp,u_yp,u_zp;            //physical velocities
  float f[Nc],ftemp[Nc],feq[Nc];
  float rho;
  Node** neighbours[18];
  bool boundary, dead;
  void create_Node (int, int, int);
  void collide ();
};
```

int i,j,k

These are the discrete lattice coordinates in which the Node stands.

These values are needed to create the list of neighbouring nodes.

float x,y,z

These are the physical coordinates of the point in space which the node represents.

float u,u_x,u_y,u_z

Discrete velocities of a node. `u` stands for the modulus of $\vec{u} = \sqrt{u_x^2 + u_y^2 + u_z^2}$

float up,u_xp,u_yp,u_zp

The discrete velocities of the node. `up` stands for the modulus of $\vec{u_p} = \sqrt{u_{xp}^2 + u_{yp}^2 + u_{zp}^2}$

float f[Nc],ftemp[Nc],feq[Nc]

The distribution function f and equilibrium distribution function f^{eq} fields are stored in one dimensional arrays of Nc dimension, where Nc stands for the number of discrete velocities of the model (in this case 19 for D3Q19). The field `ftemp` stands for a temporal storage of the f field to avoid erase distribution functions values during the streaming step.

float rho

Stores the value of the node density.

Node neighbours[18]**

This is a list of pointers to objects of Node class which are the adjacent neighbouring nodes. This list is accessed during the main loop (*i.e.* in the streaming step).

bool boundary, dead

Booleans variables are needed as a flag to indicate certain nodal properties. Boundary flag is enabled when the node is a boundary (and contrary when it is a fluid node). On the other hand, dead flag is enabled when the node is surrounded by boundary nodes, therefore the

boundary has no interaction at all with any fluid node and it can be not used in the computation to save time and resources.

create_Node

It is the constructor of the Node class and fills every node where it is applied with the standard values for a node and for its corresponding coordinates.

collide

Applies the collision step for f_a and $a = 0...18$ at every node it is applied.

There are several lists that include pointers to node class objects. It is a manner of having the nodes classified according to their properties. The lists used are:

```
std::vector<Node*>nodes;
std::vector<Node*>stream_nodes;
std::vector<Node*>dead_nodes;
std::vector<Node*>living_nodes;
std::vector<Node*>boundary;
std::vector<Node*>liquid_nodes;
```

Since the volume of study is finite, and the limiting nodes of these volume are critical and they cannot be streamed due to the fact that they do not have as much as neighbouring nodes as any other node completely inside the finite volume. Therefore the nodes can be classified in two general groups as seen in fig A.2

nodes

This is a list of pointers which are pointing to every node created.

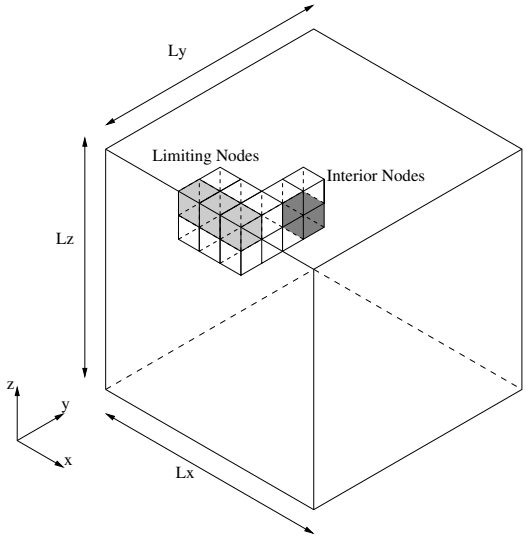


Figure A.2: Finite volume of simulation representation

stream_nodes

List of nodes that must be streamed, *i.e.* the nodes which are not dead and are interior nodes.

dead_nodes

List of pointers to nodes which are dead.

living_nodes

List of pointers to nodes that are no dead, whether they are or not limiting or interior nodes.

boundary

List of pointers to nodes that are interior nodes, and have the *boundary flag* enabled without being dead.

liquid_nodes

List of pointers to nodes that are interior nodes, and do not have the *boundary flag* enabled and of course they are not dead.

One can see that this structure provides a simple way to access to the nodes when it is required by the method by simply accessing to the lists of pointers.

Geometric classes

To implement rather complex geometries this implementation only has to be able to enable the nodal boundary flag that sets the node as a boundary node. With this field of flags the dead flag field can be also filled.
Therefore the following geometric hierarchy has been designed in order to combine different geometric forms. When only an isolated geometric form is wanted as a boundary obstacle, the class `unions` do not play any role.
Thus, this hierarchy is only used when a geometric object formed with different geometric forms combined between them is wanted. Every geometric

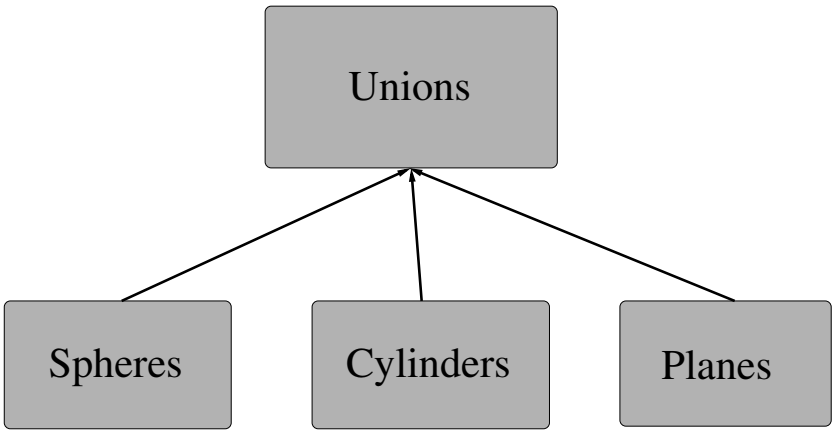


Figure A.3: Relations between geometric classes implemented

element introduced it is stored in the following pointer list as nodes.

```
std::vector<Unions*> Unionslist;
std::vector<Sphere_Obstacle*> sphere;
std::vector<Plane_Obstacle*> plane;
std::vector<Cylinder_Obstacle*> cylinder;
```

To understand better how the geometric classes can interact through the `unions` class firstly one has to understand the basic classes itself.

```
class Sphere_Obstacle{
  int x;
  int y;
  int z;
  float R;
  int full;
  int index;
  void create_sphere(float,float,float,float,int,int);
};
```

To understand the members of the sphere class one has to know that the equation that limits the geometric space occupied by a sphere is

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$$

int i,j,k

Coordinates of sphere centre (*i.e.* x_0, y_0 and z_0 in the sphere equation).

float R

Sphere radius.

int full

This is an integer flag that stand for

$$full = 1 \rightarrow (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \leq r^2 \quad \text{(A.1)}$$

$$full = 0 \rightarrow (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \geq r^2 \quad \text{(A.2)}$$

Basically it identifies if the domain of the geometric obstacle is outside or inside the sphere domain.

int index

It is an integer index that says in which `union` the sphere belongs. When `index = 999` the sphere does not belong to any union.

create_sphere

It is the sphere creator method. This method fills the sphere members with its arguments.

The plane class is a class implemented to perform cuts in the geometry with unions. It has no meaning to specify a plane alone.

```
class Plane_Obstacle{
  float a;
  float b;
  float c;
  float d;
  int full;
  int index;
  void create_plane(float,float,float,float,int, int);
};
```

A plane is defined mathematically by the following equation

$$Ax + By + Cz + D = 0$$

where the director vector perpendicular to the plane is

$$\vec{p} = (A\,, B\,, C)$$

Thus, the class members of the plane are:

float a,b,c,d

Coordinates of A,B,C and D parameters of the plane equation.

int full

This is a integer flag that stand for

$$full = 1 \rightarrow Ax + By + Cz + D \leq 0 \tag{A.3}$$

$$full = 0 \rightarrow Ax + By + Cz + D \geq 0 \tag{A.4}$$

It identifies if the nodes that are wanted to be boundaries are above or below the plane.

int index

It is an integer index that says in which `union` the plane belongs. When `index = 999` the plane does not belong to any union.

create_plane

It is the plane creator method. This method fills the plane members with its arguments.

The cylinder class follows the same structure as the sphere class. Although with this implementation only infinite cylinders can be implemented, therefore, it is mandatory to cut the cylinders with planes using a union. Also, a limitation with this geometric implementation is that only cylinders defined with *cartesian* directions can be defined because the rotation is not implemented (see Fig. A.4 for better understanding).

```
class Cylinder_Obstacle{
  float x;
  float y;
  float z;
  float r;
  int full;
  int index;
  void create_cylinder(float,float,float,float,int,int);
};
```

The cylinder equations are as follows taking into account that they describe cylinders in *cartesian* directions,

$$(1) \qquad (y - y_0)^2 + (z - z_0)^2 = r^2$$

$$(2) \qquad (x - x_0)^2 + (z - z_0)^2 = r^2$$

$$(3) \qquad (x - x_0)^2 + (y - y_0)^2 = r^2$$

They can be understood as a circumference extruded in the direction perpendicular to the plain in which the circumference is contained. It can be seen in Fig. A.4.

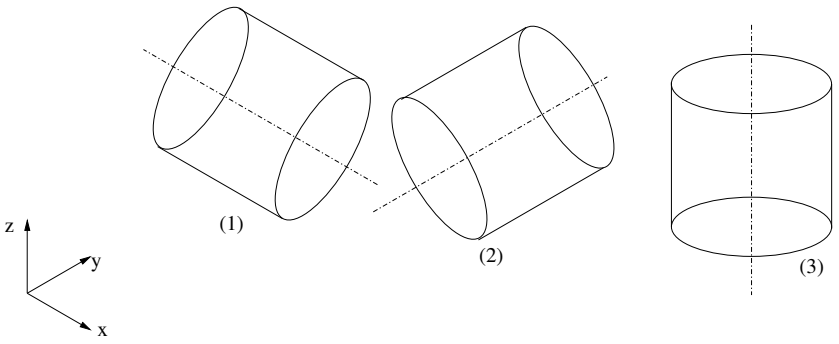


Figure A.4: Orientation permitted of the cylinders

int i,j,k

Coordinates of the centre of the circumference that fix the cylinder boundaries (*i.e.* x_0, y_0 and z_0 in the cylinder equation). Distinctively from the sphere, the cylinder equation only uses two dimensions (to define the circumference). Therefore, when one of the coordinates is null, it is understood that this is the coordinate in which the circumference is extruded.

float R

Cylinder radius.

int full

This is an integer flag that stand for

$$full = 1 \rightarrow (x - x_0)^2 + (y - y_0)^2 \leq r^2 \tag{A.5}$$

$$full = 0 \rightarrow (x - x_0)^2 + (y - y_0)^2 \geq r^2 \tag{A.6}$$

As done in the sphere, it identifies whether the domain is inside or outside the circumference extruded.

int index

It is an integer index that says in which `union` the cylinder belongs. When `index = 999` the cylinder does not belong to any union, but this is a rare case because the cylinder will be cut by planes to define its finite volume.

create_cylinder

It is the cylinder creator method. This method fills the cylinder members with its arguments.

The class `unions` is a tool used to implement interaction between the different geometric objects. It has these members

```
class Unions{
  std::vector<Plane_Obstacle*> planes;
  std::vector<Sphere_Obstacle*> spheres;
  std::vector<Cylinder_Obstacle*> cylinders;
};
```

As it can be seen this class contains as members lists of geometric objects. It is used as a *heterogeneous container* to access to its members of different classes.

Therefore, when there is a union, to define the geometry the algorithm asks for every node if it is included in every geometric object that contains the specific union. It is shown in the following *pseudocode* algorithm.

```
Data: bool is a boolean variable
Result: how unions are treated
bool ← false;
for every union do
    for every node do
        for every member of the union do
            if is inside the domain then
                bool ← true;
            else
                bool ← false;
            end
        end
        node boundary flag ← true;
    end
end
```

Algorithm 4: How unions are treated

A.1.2 Overall Program Structure

The implementation design was thought to be quite modular. Therefore the implementation can be divided in three basic modules as shown in Fig. A.5. These three components are:

1. A geometry package which reads from a file the geometry to implement
2. A data processor package that applies the algorithm
3. A data writer package which writes the data generated in several forms

The geometry package

The geometry module is the part of the implementation that is the responsible to read the geometry from a file and generate the consequent geometric objects and fills the lists of pointers to them.
It is important to notice that the geometry must be defined independently from the discrete lattice, thus it can be expressed in physical units. Hence, a simple `.txt` file is used to store the geometric objects with its properties and then it can be read by the geometry module. Obviously this `.txt` file

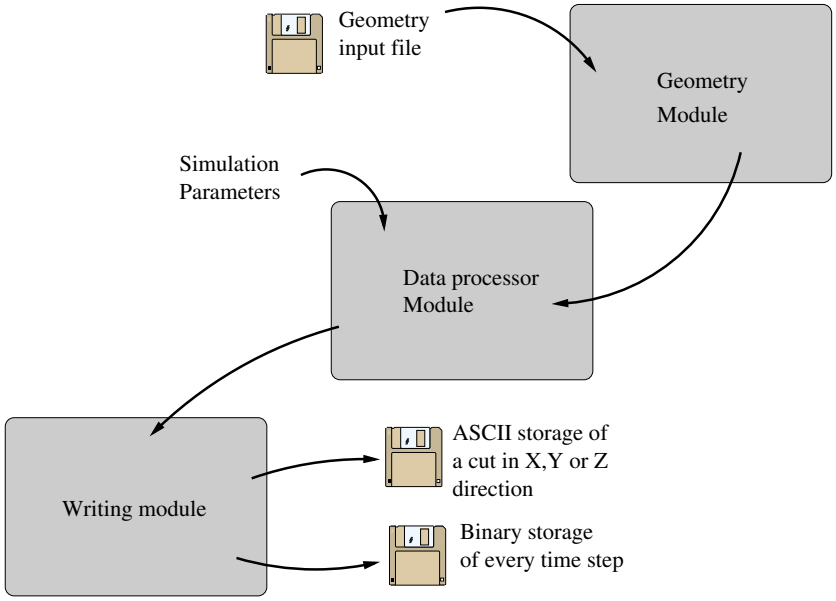


Figure A.5: Scheme of the structure of the implementation

has a syntax ,explained below, that limits its flexibilit, but it is a simple way to describe the geometry once the sintaxis is known.

```
N  X  Y  Z

cylinder
x    y    z    r    f    i
plane
a    b    c    d    f    i
sphere
x    y    z    R    f    i
```

The numbers in the first row are mandatory and they define:

N the number of geometric objects to read

X Y Z The maximum value of X, Y and Z in the finite volume in a certain unit. They are needed to scale the geometry.

One can notice that after the first row there come the different geometric objects. They have the label that marks which type of geometric object must be created, and afterwards their class members.

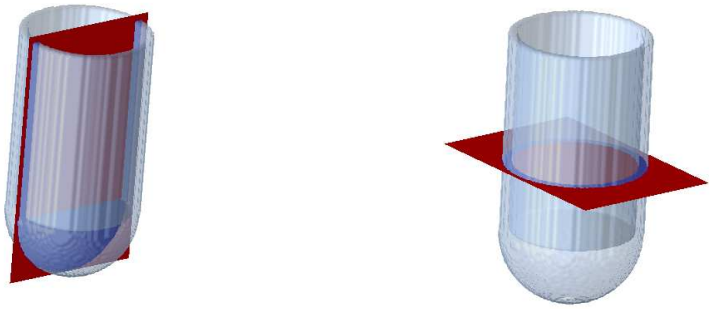
To understand the performance of this geometry module below there is an example of a geometric figure compound by different geometric objects and its unions.

```
6 8000 5500 10000

cylinder
4000 2750 0 2185 0 0
cylinder
4000 2750 0 2500 1 0
plane
0 0 1 -9800 0 0
plane
0 0 1 -2600 1 0

sphere
4000 2750 2600 2500 1 1
plane
0 0 1 -2600 0 1
```

Then, if the geometry is plotted: One can see that the geometry is basically compound by two objects. The first, which corresponds to the union with null index is the union of two concentric cylinders and two planes that cut both cylinders to limit their volume.
The other geometric object is a sphere limited by the same plane that limits the lowest part of the cylinders.



(a) geometry with a cut in $x = 0$

(b) geometry with a cut in $z = 0$

Figure A.6: 3D representation of the geometry introduced

The data processor package

This module is responsible for applying the LBM to the geometry defined previously. It has a static implementation and, hence, there are some parts that must be implemented again when the system changes its conditions. For instance, the boundary conditions must be set for every given geometry.
Further comments on the main algorithm in subsection A.1.3.
In the initialization subroutine the main steps are clear:

1. Get the simulation data from the user and dimensionalize the parameters.
2. Reserve the memory needed to store every node in the lattice.
3. Read the input data file that contains the geometry.
4. Generate the geometry in the lattice.
5. Classify the nodes in several groups (see section A.1 for further explanations)
6. Initialize the velocity of every node in the lattice.

7. Compute the equilibrium distribution function taking into account the values of the initial velocity previously initialized in step 6.

8. Store the equilibrium distribution function values calculated in step 7 in the nodal value for the distribution function.

The data writing package

As seen in Fig. A.5 the data processor module streams information to the writing module and then it stores the data in two basic formats, *ASCII* and *binary*.
The basic motivation to store the data generated is to analyze it and perform postprocessing treatments.

One of the interesting postprocessing analyses is the study of a variable field after N iterations. For this reason it has been implemented a function to store in an *ASCII* format the variable field (*i.e.* velocity field, distribution function, ρ...) for a certain instant t.
It has been chosen the *ASCII* format in front of the binary format because it is human readable, and the difference between *ASCII* or in *binary* in terms of time cost of writing a variable field for all nodes but only for one time step is negligible when the lattice dimensions are not enormous. With this implementation the user can choose after the iterations how many cuts and which ones does he want to store.

Another interesting analysis would be to study the evolution of the velocity field during the iteration time. To be able to do this analysis the velocity field must be stored in every time step. Hence, for a time-series analysis stands the first method of data storing in binary format.
It has been chosen the *binary* format instead of the *ASCII* format due to two main reasons:

1. The velocity of writing binary is much higher than writing text.

2. *ASCII* format occupies much more memory than *binary* format.

It is known that a float number occupies 4 bytes in a binary format, and every *ASCII* character occupies one byte. Therefore, if it is wanted a high precision and float variables are used to store the variables field, every float is compound by 8 digits it can be easily seen that a storing data with *ASCII* format will always require higher amounts of memory.
Supposing one wants to store a velocity field (a float number for every component) of a lattice of $N_x \times N_y \times N_z$ dimension for 10 time steps. Therefore, the required amount of memory in both formats will be:

$$\begin{aligned} Binary \rightarrow 10(N_x \times N_y \times N_z)(3 * 4\,bytes) &= 120(N_x \times N_y \times N_z)\,bytes \\ ASCII \rightarrow 10(N_x \times N_y \times N_z)(3 * 8\,bytes) &= 240(N_x \times N_y \times N_z)\,bytes \end{aligned}$$

As it can be seen, if the data uses a float format, storing in *ASCII* will always require twice the memory than the binary format.

A.1.3 Main Algorithm

The main algorithm was shown in section 2.10.
The application of the LBM is straightforward, but one must be careful in which nodes each function is applied. Therefore the algorithm 5 shows the argument of each function. See the explanation of every list in A.1.1.

A.1.4 Computing requirements

In this subsection the computer capability required to perform the LBM with this actual implementation will be studied.

Previously in subsection A.1.2 a comparison between the data storage alternatives has been made. In this subsection the study will be focused on the memory needed (*i.e.* RAM memory) to store the data used during the

```
Result: Aplication of the LBM
Initialize;
for number of iterations do
    apply boundary conditions;
    Stream Step (stream nodes);
    Compute macroscopic quantities (stream nodes);
    Compute f^eq (living nodes);
    Collision Step (living nodes);
    if there are obstacles then
        Obstacle treatment (boundary);
    end
    write time-step data in binary;
end
write data cuts in ASCII;
```

Algorithm 5: Aplication of the LBM

process to perform the LBM.

The RAM memory needed will be approximated and not exactly calculated, hence, it will be considered that only the vector lists and the lattice will be stored (neglecting the other variables because they only supose several bytes of memory in front of the other).

Therefore, first one has to define the size of each list and each variable.

vector lists

There are 6 lists of pointers to nodes (see A.1.1). Supposing that the *boundary* list, and thus the *dead_nodes* list are null because in the lattice there is no node which is a boundary, there are still 4 lists of pointers.

Assuming valid that the approximation in which the lists have the same size (they do not have, but the difference between them is small) there are lists of $(N_x \times N_y \times N_z)$ pointers. Each pointer in a 32-bit architecture occupies 4 bytes (32 bits). Therefore

$$lists\ size\ = \ 4\,(N_x \times N_y \times N_z) \times 4\,bytes$$

$$lists\ size\ = 16\,(N_x \times N_y \times N_z)\,bytes$$

One can see that for rather large lattices can have more than $200 \times 200 \times 200$ nodes. Then, the list size increases considerably to Mbytes magnitude (128MB in this exact case), but taking into account the general specifications of the computers nowadays, a personal computer can handle this requirements without any problem.

Node class

Every node class object created is stored in the RAM memory to be able to be accessed during the program execution. In subsection A.1.1 can be seen how a node class object is compound. Therefore every node class object occupies:

$3 \times int$	$=3 \times 4\ bytes$	$=12\ bytes$
$12 \times float$	$=12 \times 4\ bytes$	$=48\ bytes$
$3 \times float[19]$	$=3 \times 19 \times 4\ bytes$	$=228\ bytes$
$1 \times pointer[18]$	$=1 \times 18 \times 4\ bytes$	$=72\ bytes$
$2 \times bool$	$=2 \times 1\ byte$	$=2\ bytes$

Hence, the total amount of memory required for a single node class object is

$$Total\ size\ = 362bytes$$

If the lattice has $N_x \times N_y \times N_z$ dimensions, the amount of RAM memory needed to execute the algorithm will be $N_x \times N_y \times N_z \times 362\ bytes$. Thus, if a lattice of $200 \times 200 \times 200$ is created, the execution requires 2896 MB to store the lattice in addition to the 128MB needed to store the list of pointers. Therefore, to execute rather large lattices a cluster is needed.

Bibliography

[AEM09] Usman R. Alim, Alireza Entezari, and Torsten MÃűller. The lattice-boltzmann method on optimal sampling lattices. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):630–641, 2009.

[BG00] J. M. Buick and C. A. Greated. Gravity in a lattice Boltzmann model. *Physical Review E*, 61(5):5307–5320, 2000.

[bil] Bilineal interpolation. http://es.wikipedia.org/wiki/Interpolaci

[Bui97] J. M. Buick. *Lattice Boltzmann methods in interfacial wave modelling*. PhD thesis, The University of Edinburgh, 1997.

[C] C++ webpage. http://www.cplusplus.com/.

[Cai05] A. Caiazzo. Analysis of lattice Boltzmann initialization routines. 2005.

[CFHL09] Kuen-Hau Lin Chih-Fung Ho, Cheng Chang and Chao-An Lin. Consistent boundary conditions for 2d and 3d lattice boltzmann simulations. 2009.

[CK04] I.M. Cohen and P.K. Kundu. *Fluid Mechanics*. Elsevier Science, 2004.

[DX] Opendx webpage. http://www.opendx.org/.

[Eng] Simon T. Engler. Benchmarking the 2D Lattice Boltzmann BGK Model. *Complex Simulation Report*.

[GNGB97] Martha A. Gallivan, David R. Noble, John G. Georgiadis, and Richard O. Buckius. An evaluation of the bounce-back boundary condition for lattice boltzmann simulations. *International Journal for Numerical Methods in Fluids*, 25(3):249–263, 1997.

[GNU] Gnuplot webpage. http://gnuplot.sourceforge.net/.

[Gom94] T.I. Gombosi. *Gaskinetic Theory*. Cambridge Atmospheric and Space Science Series. Cambridge University Press, 1994.

[HH10a] Martin Hecht and Jens Harting. Implementation of on-site velocity boundary conditions for d3q19 lattice boltzmann simulations. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(01):P01018, 2010.

[HH10b] Martin Hecht and Jens Harting. Implementation of on-site velocity boundary conditions for d3q19 lattice boltzmann simulations. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(01):P01018, 2010.

[HL97] Xiaoyi He and Li-Shi Luo. A priori derivation of the lattice boltzmann equation. *Phys. Rev. E*, 55:R6333–R6336, Jun 1997.

[IMLF09] Salvador Izquierdo, Paula Martínez-Lera, and Norberto Fueyo. Analysis of open boundary effects in unsteady lattice boltzmann simulations. *Computers & Mathematics with Applications*, 58(5):914 – 921, 2009. Mesoscopic Methods in Engineering and Science.

[Lat08] Jonas Latt. How to chose lattice units in a LB simulation. Technical report, May 2008.

[Lut06] James F. Lutsko. Chapman-enskog expansion about nonequilibrium states with application to the sheared granular fluid. *Phys. Rev. E*, 73:021302, Feb 2006.

[Mel13] Igor Mele. Lattice boltzmann method. 2013.

[Moh11] A.A. Mohamad. *Lattice Boltzmann Method: Fundamentals and Engineering Applications with Computer Codes*. SpringerLink : Bücher. Springer, 2011.

[Pen] Chen Peng. The lattice boltzmann method for fluid dynamics: Theory and applications. Master's thesis, École Polytechnique Fédérale de Lausanne.

[Sat10] A. Satoh. *Introduction to Practice of Molecular Simulation: Molecular Dynamics, Monte Carlo, Brownian Dynamics, Lattice Boltzmann and Dissipative Particle Dynamics*. Elsevier insights. Elsevier Science, 2010.

[Sch10] Martin Schreiber. Gpu based simulation and visualization of fluids with free surfaces. Diplomarbeit, Institut für Informatik, Technische Universität München, June 2010.

[SJ06] Michael C. Sukop and Daniel T. Thorne Jr. *Lattice Boltzmann Modeling - An Introduction for Geoscientists and Engineers*. Springer, Berlin, 2006.

[SYJ+13] SungWan Son, HyunSik Yoon, HaeKwon Jeong, ManYeong Ha, and S. Balachandar. Discrete lattice effect of various forcing methods of body force on immersed boundary-lattice boltzmann method. *Journal of Mechanical Science and Technology*, 27(2):429–441, 2013.

[Thu03] Nils Thuerey. A Lattice Boltzmann method for single-phase free surface flows in 3D. *Master thesis*, 2003.

[Thu07] N. Thuerey. Physically based Animation of Free Surface Flows with the Lattice Boltzmann Method. *PhD thesis*, ISBN 978-3-89963-519-5, Mar 2007.

[TR09] Nils Thürey and Ulrich Rüde. Stable free surface flows with the lattice boltzmann method on adaptively coarsened grids. *Computing and Visualization in Science*, 12(5):247–263, 2009.

[Wika] Wikipedia. Hagen-poiseuille-law. http://en.wikipedia.org/wiki/Hagen

[Wikb] Wikipedia. Pwr image. http://en.wikipedia.org/wiki/File:Reactorvessel.gif.

[ZH98] Q Zou and X He. On pressure and velocity flow boundary conditions for the lattice boltzmann bgk model. Technical Report comp-gas/9508001. LA-UR-95-2708, Los Alamos Nat. Lab., Los Alamos, NM, Dec 1998.